DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO

Algorithms for Representing Similarity Data

Michael D. Lee

DSTO-RR-0152

ALGORITHMS FOR REPRESENTING SIMILARITY DATA

Michael D. Lee

Communications Division
Electronics and Surveillance Research Laboratory

DSTO–RR–0152

ABSTRACT

This report develops and demonstrates algorithms for representing and displaying similarity data using three established cognitive models. The first representational model, multidimensional scaling, represents objects as points in a coordinate space so that similar objects lie near each other. The second representational model, the additive tree, represents objects as terminal nodes in a tree so that the similarity of two objects is modelled by length of the path between them. The third representational model, additive clustering, specifies a number of clusters with associated weights, so that the similarity of two objects is modelled by the sum of the weights of their common clusters. As well as listing and demonstrating MATLAB algorithms for finding these representations, a survey is presented of ways in which similarity and proximity data may be generated, and a principled Bayesian method of controlling the complexity of each representational model is presented. Finally, a number of suggestions are made regarding the use of the three representational models, and the relative strengths and weaknesses of the algorithms in relation to previously developed alternative algorithms are discussed.

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury, South Australia, Australia 5108

Telephone: (08) 8259 5555
Facsimile: (08) 8259 6567

AR No. AR–010–971
May, 1999

APPROVED FOR PUBLIC RELEASE

Algorithms for Representing Similarity Data

EXECUTIVE SUMMARY

In many defence-related contexts, it is important to be able to form *models* of domains of interest. Models provide a mechanism for both comprehension and prediction. They give observed events a meaningful interpretation, and allow future or unseen events to be anticipated through a process of generalisation. Ideally, by using appropriate domain models, commanders can initiate pre-emptive manoeuvres in relation to predicted enemy behaviour, surveillance systems can allocate limited resources to those areas most likely to yield useful data, intelligence analysts can develop a high-level understanding of the behaviour of their targets, and so on.

A key requirement for the development of a good domain model is the construction of a good underlying domain *representation.* Representations are the stable components of domain models, specifying the nature of the objects in the domain, and their relationships to each other. A good representation should consist of a simple but accurate description of this domain structure. It should provide both the explanatory insight needed for understanding, and the conceptual basis from which predictions can be made when processes are added to form a complete model. One practical way of developing these sorts of accounts is by developing sophisticated cognitive representations of similarity data collected within domains of interest.

Representations based on similarity or proximity data are particularly applicable in domains where either an unmanageably large number of raw measurements must be understood, or in domains where it is impossible to make valid and reliable direct measures of the variables of interest, and it is necessary to rely on subjective human judgments. In the first case, building representations based on similarity can lead to a simple but accurate summary representation of the available data. In the second case, a sophisticated representation of the similarity judgments can be used to reveal the underlying variables of interest.

Accordingly, this report provides a survey of ways in which similarity or proximity data may be produced for a wide variety of domains, and considers three established cognitive representational approaches that are applicable to this sort of data. These approaches—multidimensional scaling, additive trees, and additive clustering—constitute different, but largely complementary, forms of knowledge representation. In addition, their empirical success as models of human mental representation gives them the advantage of being particularly well suited for human comprehension and analysis.

Algorithms for generating each type of representation are developed and demonstrated, together with examples and algorithms for displaying the representations. A particular emphasis is given to the need to make the representations as simple as possible, and a formal approach to choosing the representation that best balances the competing demands of simplicity and accuracy is presented. Through a series of case studies, it is shown that this approach to representational development leads to the generation of domain representations that are readily amenable to substantive interpretation, and could potentially form the basis of predictive domain models. Finally, the relative strengths and weaknesses of

alternative algorithms for generating the three types of representations are discussed, and some suggestions are made regarding the sorts of situations for which each representational type is the best suited.

Authors

Michael D. Lee
Communications Division

Michael Lee completed a Bachelor of Science in the Faculty of Mathematical Sciences between 1988–1990, Honours in the Department of Psychology in 1992, and a Ph.D. jointly supervised within the Departments of Psychology and Electrical & Electronic Engineering between 1994–1996, all at the University of Adelaide. During the final year of Ph.D. candidature, he was a postgraduate member of the Cooperative Research Centre for Sensor Signal & Information Processing. In January 1997 he commenced employment with what is now known as the Intelligent Networks Group, in the Secure Communications Branch of the Communications Division at the DSTO.

His research interests are in the areas of cognitive modelling and artificial intelligence, particularly in relation to models of human mental representation.

THIS PAGE IS INTENTIONALLY BLANK

Contents

1 Introduction

One general and useful way of describing a domain of interest is to identify a set of objects in that domain, and then provide a scalar measure of the *similarity* or *proximity* between each pair of those objects. Social networks, for example, may be thought of in terms of a set of actors, and the various strengths of the relationships between them. A library may be conceived as consisting of a set of information items that have different degrees of semantic relatedness. Sporting competitions may be seen to be made up of a number of teams that have various likelihoods of defeating each other when they play.

It is usually relatively straightforward to characterise domains of interest in this way—all that is required is the quantification of 'local' relationships between pairs of objects. Collections of these local observations, however, often implicitly encode a wealth of 'global' information about the structure of the domain as a whole. The relationships between pairs of actors in a social network, for example, may reveal the existence of larger social groupings, such as cliques. The semantic relatedness of library items may indicate a number of loosely grouped clusters, corresponding to different subjects or themes. The likelihood of different sporting teams defeating each other may allow inferences to be made regarding an overall ranking, or 'ladder', including all of the teams. The usefulness of characterising domains in terms of local similarity or proximity measures stems from the possibility of being able to reveal these sorts of global domain properties.

This report develops and demonstrates algorithms[1] for generating a global representation of a set of objects from local similarity or proximity measures. These algorithms attempt to fit the data to three representational models known as multidimensional scaling, additive trees, and additive clustering. All of these representational approaches were originally developed within cognitive psychology as models of human mental representation [66], but are amenable to far wider application. Indeed, each of the representational models may be regarded as a general pattern recognition or data modelling technique. Because of their psychological origins, however, they tend to offer a fresh perspective on common representational problems. For this reason, the approaches presented here are well worth considering for application in a wide variety of representational modelling problems.

The first approach, *multidimensional scaling*, represents objects as points in a coordinate space. These points are positioned so that the proximity of two objects is modelled by the distance between them, as measured according to some distance metric. An example of the multidimensional scaling representation for 5 objects, together with its associated proximity matrix, is presented in Figure 1

The second approach, an *additive tree*, represents objects by terminal (leaf) nodes in a tree. A set of lengths are also assigned to each of the arcs in the tree, so that the proximity of two objects is modelled by the sum of the lengths of the unique path between them. An example of an additive tree and the proximity matrix it represents, based on an example given in [16], is presented in Figure 2.

The final representational approach, *additive clustering*, specifies a number of clusters, each of which is assigned a weight, so that the similarity of two objects is modelled by

[1]The algorithms, and sample similarity data, are available on the internal DSTO Web at the URL http://www-203.dsto.defence.gov.au/~mdl

	A	B	C	D	E
A	--				
B	12	--			
C	20	16	--		
D	16	20	12	--	
E	10	10	10	10	--

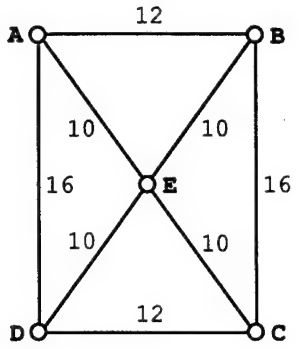

Figure 1: An example of an multidimensional scaling representation and its associated proximity matrix.

	A	B	C	D	E
A	--				
B	15	--			
C	20	25	--		
D	18	23	06	--	
E	20	25	20	18	--

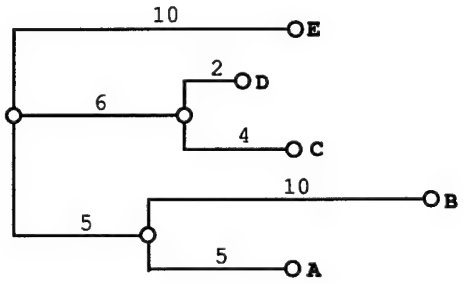

Figure 2: An example of an additive tree representation and its associated proximity matrix.

	A	B	C	D	E
A	--				
B	17	--			
C	17	17	--		
D	05	05	14	--	
E	05	05	05	05	--

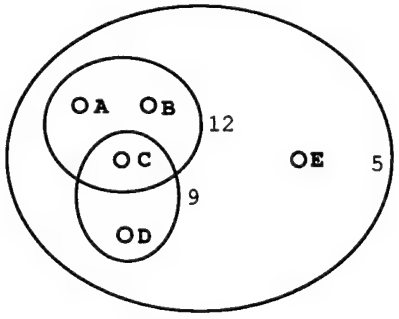

Figure 3: An example of an additive clustering representation and its associated similarity matrix.

the sum of the weights of the clusters to which both belong. Additive clustering places no constraints upon the construction of the clusters, allowing them to overlap in arbitrary ways. An example of an additive clustering model with 3 clusters, together with the similarity matrix it represents, is shown in Figure 3.

These three representational models are largely complementary, with each having different strengths and weaknesses. For this reason, the relative usefulness of their application will typically be domain specific, and depend on both the nature of the similarity or proximity data, and the over-arching goals of the representational modelling. This issue is addressed further in the Discussion.

2 Generating Similarity and Proximity Data

Multidimensional scaling, additive tree, and additive clustering representations are all derived from matrices of similarity or proximity. A similarity matrix $\mathbf{S} = [s_{ij}]$ details the similarity between all $n(n-1)/2$ distinct pairs of objects, while a proximity matrix $\mathbf{D} = [d_{ij}]$ gives a distance-like proximity measure between each pair. Both similarity and proximity matrices should be symmetric and contain non-negative entries, and it is often convenient to normalise the entries to lie between 0 and 1.

The concepts of similarity and proximity are obviously related, in the sense that larger measures of similarity correspond to smaller proximities or distances. This means that similarity matrices may readily be converted into proximity matrices, and vice versa. In general, any monotonically decreasing function could be used, although the simplicity of the linear transformation $s_{ij} = \gamma - d_{ij}$ makes it appealing. As an alternative, it should be noted that a compelling argument has been developed [67, 69] that human cognition relies on an exponentially decaying transformation of the form $s_{ij} = \exp(-\gamma d_{ij})$.

Sometimes, the set of objects for which a representation is sought may already be represented by a symmetric similarity or proximity matrix. More often, however, it will be necessary to form such matrices from 'raw' representational information that describes the objects in some other way. Perhaps the two most common possibilities involve the original representation taking the form of a non-symmetric matrix of broadly 'associational' measures, or the objects initially being represented in terms of their values across a list of properties.

2.1 Measures Of Association

In some circumstances, particularly in experimental studies, objects may initially be represented in terms of the likelihood of confusion of pairs of objects, their probability of co-occurrence, their frequency of substitution, a set of similarity or dissimilarity ratings provided by observers, or any of a range of other plausible indicators of general association.

Given a matrix of pairwise associational measures, denoted $\mathbf{A} = [a_{ij}]$, arising from any of these sources, it is possible simply to generate a similarity matrix by transpose averaging, so that the similarity between the ith and jth objects is given by:

$$s_{ij} = \frac{a_{ij} + a_{ji}}{2}. \tag{1}$$

A more principled [61] approach, particularly when dealing with probabilities, is given by the transformation:

$$s_{ij} = \sqrt{\frac{a_{ij}a_{ji}}{a_{ii}a_{jj}}}, \tag{2}$$

while a third option, which may have some practical advantages in terms of robustness [64], takes the form:

$$s_{ij} = \frac{a_{ij} + a_{ji}}{a_{ii} + a_{jj}}. \tag{3}$$

An example of this general approach is found in the generation of a similarity matrix [64] from experimental data detailing the probability of auditory confusion for 16 consonant phonemes. The original data [45] took the form of confusion matrices for each of 6 levels of signal to noise ratio. To form a similarity matrix, each of these confusion matrices was transformed using Equation 3, and the results averaged.

2.2 Lists Of Properties

Often objects are initially represented by a vector giving the values they assume across a number of quantitative properties. Analysis of these sorts of domains is usually approached by retaining the structure provided by the vectors, and examining the distributions of the variables, the various relationships and dependencies between them, and so on.

There are, however, a number of advantages in using the property lists to generate a similarity or proximity matrix, and then deriving a multidimensional scaling, additive tree, or additive clustering model. There is no guarantee that the representational structure imposed by the vector description of the objects constitutes an optimal, or even an effective, framework within which to understand the domain. That is, descriptions based on 'intuitively reasonable' or 'canonical' domain properties may fail to identify the fundamental latent variables of interest in domain analysis. Applying an understanding of the objects and their domain to generate similarity or proximity measures affords an opportunity to circumvent this sort of arbitrariness in a given characterisation. In addition, even when the provided lists of properties are appropriate, the generation of multidimensional scaling, additive tree, and additive clustering representations has natural application in the context of data visualisation, as demonstrated by the survey provided in [42]. Of course, there are also potential disadvantages in representing objects originally described in terms of lists of properties through a mediating similarity or proximity matrix, particularly in terms of the computational burden involved but, in general, the approach has merit in many applied situations.

One way to generate a proximity matrix from lists of properties is by applying any distance measure to the representational vectors describing the objects. Many familiar

distance measures, such as the Euclidean and Hamming metrics, may be considered as a special case of the family of weighted Minkowskian distance metrics. Under this approach, given two property vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, the proximity between the ith and jth objects is given by:

$$d_{ij} = \left[\sum_k w_k \left| v_{ik} - v_{jk} \right|^r \right]^{\frac{1}{r}}, \tag{4}$$

where w_k is the weight given to the kth property, and r is a parameter that determines which of the family of metrics is employed. The choice $r = 2$ corresponds to the Euclidean metric, $r = 1$ gives what is variously described as the City-Block, Taxi-Cab or Manhattan metric, while as $r \to \infty$ the Supremum or Maximum metric is approximated. Despite its generality, the weighted Minkowskian family obviously does not encompass all possibilities. Other distance metrics, not subsumed by this family, such as the Mahalanobis, Canberra, Bray-Curtis, Bhattacharyya and entropy distances could also be employed to generate proximity matrices [18].

As an example, consider the problem of generating a proximity matrix for a set of countries, based on a list of quantitative economic indicators for those countries. Thus, for each country, the raw information takes the form of a list of the GDP, unemployment rate, inflation rate, and so on. One approach would be to measure the proximity between each pair of countries as the City-Block distance between the vectors containing these measures. Effectively, this proximity would be a sum of the differences between the countries across each of the measures. A weighted City-Block metric would allow a weight to be associated with each of the indicators, so that each could be normalised, or greater emphasis could be placed on those indicators deemed to be more important.

For an example of a proximity measure not based on the weighted Minkowskian metric, consider a set of random variables, perhaps describing the frequency characteristics of a set of signals. Using the entropy distance measure for a pair of random variables, defined as the difference between their joint entropy and mutual information [44], the construction of a proximity matrix is straight-forward.

Similarity matrices can also be generated from descriptions of objects in terms of their values across a list properties. General possibilities include the weighted dot- or inner-product:

$$s_{ij} = \sum_k w_k v_{ik} v_{jk}, \tag{5}$$

its normalised form as a measure of angular separation:

$$s_{ij} = \frac{\sum_k w_k v_{ik} v_{jk}}{\left[\sum_k w_k v_{ik}^2 \sum_k w_k v_{jk}^2 \right]^{\frac{1}{2}}}, \tag{6}$$

or a weighted measure of correlation:

$$s_{ij} = \frac{\sum_k w_k \left(v_{ik} - \bar{\mathbf{v}}_i\right)\left(v_{jk} - \bar{\mathbf{v}}_j\right)}{\left[\sum_k w_k \left(v_{ik} - \bar{\mathbf{v}}_i\right)^2 \sum_k w_k \left(v_{jk} - \bar{\mathbf{v}}_j\right)^2\right]^{\frac{1}{2}}}. \tag{7}$$

One example of this approach is found in established n-gram measures of orthographic[2] similarity for textual documents [19]. The 'raw' information available for a text document takes the form of a vector of frequency counts for each sequence of n consecutive characters that occurs in the document. One way in which the similarity of two documents is measured from these document vectors is through calculating their dot-product [19, 12].

Another example, using a correlational measure of similarity, is given by the study of psychoactive drug use reported in [31]. In this study, a similarity matrix for 13 drugs was formed by calculating correlations from raw data that indicated the frequency, on a 5-point scale, of the use of the drugs across a large number of students. In this way, a pair of drugs was deemed to be similar if students reported similar patterns for the frequency of use of those drugs.

In the special case that each of the properties describing the objects is a binary variable, at least a dozen additional methods for generating similarity measures have been proposed [18]. Perhaps the most important of these measures are the Jaccard coefficient:

$$s_{ij} = \frac{\sum_k v_{ik} v_{jk}}{\sum_k v_{ik} v_{jk} + \sum_k v_{ik}\left(1 - v_{jk}\right) + \sum_k \left(1 - v_{ik}\right) v_{jk}}, \tag{8}$$

and the Matching coefficient:

$$s_{ij} = \frac{\sum_k v_{ik} v_{jk} + \sum_k \left(1 - v_{ik}\right)\left(1 - v_{jk}\right)}{\sum_k v_{ik} v_{jk} + \sum_k v_{ik}\left(1 - v_{jk}\right) + \sum_k \left(1 - v_{ik}\right) v_{jk} + \sum_k \left(1 - v_{ik}\right)\left(1 - v_{jk}\right)}. \tag{9}$$

As an example, consider a social network, in which the raw data takes the form of a binary indication of whether or not each pair of actors have a relationship. Although this data could itself be used as a similarity matrix, a potentially more informative matrix may be derived using either Jaccard or Matching coefficients. Under the Jaccard approach, the similarity of a pair of actors is determined by the number of other actors with whom they both have a relationship, and normalised by a measure of their total number of relationships. Using the Matching coefficient, similarity is assigned when both actors either do, or do not, have a relationship with another actor. In either case, a much richer similarity measure is derived by taking account of all of the relationships of both actors, rather than just whether or not they have a relationship with each other.

When some of the properties have nominal or qualitative levels of data scaling, it is necessary to define uni-dimensional similarities or proximities between each possible pair of values for these properties. This may be done by prescribing some sort of rule that assigns a similarity or proximity measure to a given pair of objects, or through the exhaustive specification of appropriate values for each possible pairing.

[2] and putative semantic

For example, a rule for determining the similarity of a set of countries might be stated as follows: If the two countries are part of the same continent and speak the same language, then their similarity is 1. If they speak the same language, but are part of different continents, then their similarity is 0.6. If they are part of the same continent but speak different languages, their similarity is 0.3, and if they have neither language nor continent in common, then their similarity is 0.1. Although it is usually possible to provide sensible definitions of this type, they clearly are highly domain and property dependent.

3 Algorithms

3.1 Algorithmic Preliminaries

Before describing the multidimensional scaling, additive tree and additive clustering algorithms, it is worth giving a more general account of two optimisation approaches used in the algorithms themselves. The first of these, the pinning approach to gradient descent based optimisation, is used in the multidimensional scaling and additive tree display algorithms to find spatial coordinates for objects. The second, Population-Based Incremental Learning, is used in the additive tree generation and additive clustering algorithms to find discrete-valued representations that optimise some evaluative criterion.

3.1.1 The Pinning Variant of Gradient Descent Optimisation

The pinning variant of gradient descent optimisation [21] addresses the problem of finding a set of parameters $(\theta_1, \theta_2, \ldots, \theta_k) \in \mathbf{R}^k$ that (locally) minimises a sum squared error function $\mathrm{SSE}(\theta_1, \theta_2, \ldots, \theta_k)$, and basically works as follows: on each iteration, each of the parameters is, in a random order, temporarily fixed or 'pinned' to its current value. The values of each of the other parameters are then moved against the gradient of the error with respect to the pinned parameter, but without regard to interactions with other parameters.

Formally, if the sum squared error is re-written as:

$$\begin{aligned} \mathrm{SSE}(\theta_1, \theta_2, \ldots, \theta_k) &= \sum_{i<j} (x_{ij} - \hat{x}_{ij}(\theta_i, \theta_j))^2 \\ &= \frac{1}{2} \sum_i \sum_{j \neq i} (x_{ij} - \hat{x}_{ij}(\theta_i, \theta_j))^2 \\ &= \frac{1}{2} \sum_i \sum_{j \neq i} E_{ij}(\theta_i, \theta_j), \end{aligned} \tag{10}$$

then, when the ith parameter is pinned, the values of all of the other parameters are changed according to the rule:

$$\theta_j^{\mathsf{new}} = \theta_j^{\mathsf{old}} - \lambda \frac{\partial E_{ij}}{\partial \theta_j} \qquad \forall j \neq i, \tag{11}$$

where $0 < \lambda \leq 1$ is the learning rate.

3.1.2 Population-Based Incremental Learning

For the more difficult combinatorial optimisation problems that need to be solved to generate additive tree and additive clustering representations, the 'black-box' technique known as Population-Based Incremental Learning (PBIL) [2] is used. The basic approach of PBIL is to encode potential solutions to an optimisation problem in terms of a bit string, and maintain and update an explicit measure, in a vector π, of the (unconditional) probabilities describing the state of each of these bits in good solutions. On each iteration, a set of potential solutions $\{\xi_i\}$ is stochastically generated according to π, and evaluated against the optimisation problem at hand. The best k of these ordered solutions, $\{\xi_1, \xi_2, \ldots, \xi_k\}$ are then used to update π, using the standard competitive learning rule [30]:

$$\pi^{\mathsf{new}} = (1 - \lambda)\,\pi^{\mathsf{old}} + \lambda \xi_i \quad \text{for each } i = 1, \ldots, k\,, \tag{12}$$

where $0 < \lambda \leq 1$ is again a learning rate[3]. Each element of the probability vector π then has a small probability, μ^{prob}, of being subjected to 'mutation', which involves an additive shift, μ^{shift}, towards the maximum entropy value of 0.5. The next iteration then commences, generating another set of candidate solutions according to the updated probability vector. During these iterations a record is maintained of the best solution vector found, and the algorithm terminates once a fixed number of evaluations have been made without improvement to this best solution.

PBIL has been shown to be an effective optimisation technique for a wide range of problems, including the travelling salesman problem, bin packing, graph colouring, and general function optimisation [2, 3]. The obvious requirement for its application, however, is that potential solutions must be encoded as binary strings.

3.2 Multidimensional Scaling

The multidimensional scaling algorithm, as with all of the algorithms developed here, takes the form of a MATLAB [73] function written as a .m-file, called `mds.m`, which is listed in Appendix A. The basic approach of the algorithm is to find an $n \times m$ matrix $\mathbf{P} = [p_{ij}]$ containing the coordinates for n objects in an m-dimensional space, so that the matrix of pairwise distances between these points $\hat{\mathbf{D}} = [\hat{d}_{ij}]$ approximates the given proximity matrix $\mathbf{D} = [d_{ij}]$. The distance between the ith and jth points is measured according to one of the Minkowskian r-metrics:

$$\hat{d}_{ij} = \left[\sum_{k=1}^{m} |p_{ik} - p_{jk}|^r \right]^{\frac{1}{r}}, \tag{13}$$

as determined by the given value of r.

[3]While it is true that the competitive learning rule is asymmetric, it is not ambiguous. As implemented and evaluated in [2, 3], the 'best' vectors are applied strictly in order of their quality as potential solutions.

When deciding which of the Minkowskian metrics is appropriate, it is often informative to consider the domain in terms of established psychological notions of stimulus structure. There is a prominent approach in cognitive modelling, summarised in [52], that places particular emphasis on the $r = 1$ (City-Block) and $r = 2$ (Euclidean) choices because of their relationship, respectively, to so-called 'separable' and 'integral' stimulus domains [26, 68]. Integral stimuli are those, such as colours, that are relatively unanalyzable, in the sense that they are not readily perceived in terms of their component dimensions. Separable stimuli, in contrast, are those in which a number of component dimensions can be considered independently, such as a set of geometric stimuli varying in size and shape. Empirically, integral stimulus dimensions may be identified by testing for filtering interference, whereby performance in attending to one dimension is affected by the other, and redundancy gains, whereby performance on one dimension is facilitated by redundant information on the other.

More generally, it has been argued [68] that the distinction between separable and integral stimuli may represent endpoints of a continuum rather than a dichotomy. In particular, some stimulus dimensions satisfy the filtering interference but not the redundancy gains criterion for integrality, and are sometimes termed 'configural' dimensions. It seems likely that domains containing stimuli of this type may be modelled appropriately using Minkowski r-metrics with an r value between 1 and 2. Although values of r greater than 2 are sometimes considered [35, 68], it is difficult to provide a psychological interpretation, in terms of component structure, for stimuli modelled in this way. Pure integrality at $r = 2$ would seem to constitute a psychological upper limit on the degree to which underlying stimulus dimensions may be combined. In contrast, the adoption of metrics with $r < 1$ has been given a psychological justification [27, 67, 68] in terms of modelling stimuli with component dimensions that 'compete' for attention. It seems reasonable, therefore, to conclude that there is some psychological impetus for restricting the family of Minkowski r-metrics to the range $0 < r \leq 2$. Interestingly, some computational corroboration of this assertion may be found in evidence that, for every metric with $r > 2$, there is another 'quasi-equivalent' metric with $r < 2$ that is capable of accommodating a spatial representation with essentially the same level of error [4, 65]. In this sense, the introduction of metrics with $r > 2$ does not afford representational possibilities not available using $0 < r \leq 2$. It is, however, necessary to restrict attention further to the interval $1 \leq r \leq 2$ to preserve the metric structure of a multidimensional scaling representation, in the sense of satisfying the metric axioms.

Whatever Minkowskian metric is chosen, the `mds.m` algorithm applies the pinning variant of gradient descent optimisation to the sum squared error measure:

$$\mathrm{SSE} = \frac{1}{2} \sum_i \sum_{j \neq i} \left(d_{ij} - \hat{d}_{ij} \right)^2, \tag{14}$$

which, after differentiation, leads to the learning rule:

$$p_{jk}^{\mathsf{new}} = p_{jk}^{\mathsf{old}} + \lambda \left(d_{ij} - \hat{d}_{ij} \right) \hat{d}_{ij}^{1-r} \left| p_{ik} - p_{jk} \right|^{r-1} \operatorname{sgn}\left(p_{ik} - p_{jk} \right) \; \forall j \neq i, \tag{15}$$

where $\operatorname{sgn}(\cdot)$ is the signum function.

The algorithm requires as input a square and symmetric proximity matrix and the dimensionality of the representational space, and returns the coordinate locations of a solution, and a measure of the quality of this solution. Unfortunately, the final sum squared error measure is not scale invariant and, consequently, is often difficult to interpret. Accordingly, a more meaningful measure giving the variance of the proximity data accounted for by the solution:

$$\mathrm{VAF} = 1 - \frac{\sum_{i<j}\left(d_{ij} - \hat{d}_{ij}\right)^2}{\sum_{i<j}\left(d_{ij} - \bar{d}\right)^2} = 1 - \frac{\mathrm{SSE}}{\sum_{i<j}\left(d_{ij} - \bar{d}\right)^2}, \tag{16}$$

is provided. The algorithm assumes the operation of the Euclidean metric ($r = 2$) by default, but allows any of the Minkowskian family to be specified. Similarly, the value of the learning rate λ, and the number of iterative updates of the coordinate locations, are both assigned sensible default values that can be over-ridden if necessary.

The following is the transcript of a sample MATLAB session using the multidimensional scaling algorithm to recover the representation shown in Figure 1 from the given proximity matrix:

```
>> help mds

  MDS multidimensional scaling (michael.d.lee@dsto.defence.gov.au)
  [points,vaf]=mds(distance,dimensions,metric,iterations,learnrate)

  DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
  DIMENSIONS specifies the required dimensionality of the coordinate representation (required)
  METRIC specifies the Minkowski distance metric operating in the space (default=2)
  ITERATIONS specifies the number of optimisation iterations performed (default=50)
  LEARNRATE specifies the learning rate used in optimisation (default=0.05)

  POINTS returns an NxDIMENSIONS matrix giving the derived coordinate locations
  VAF returns the variance of the distance values accounted for by the solution

>> d=[00 12 20 16 10;
      12 00 16 20 10;
      20 16 00 12 10;
      16 20 12 00 10;
      10 10 10 10 00];

>> [points,vaf]=mds(d,2,2,100,.05)
points =
   -0.0697    9.9256
    9.5443    2.7450
   -0.0037  -10.0742
   -9.6368   -2.9192
   -0.0622   -0.0653
vaf =
    1.0000

>>
```

Some theoretical arguments and empirical comparisons, examining both the computational efficiency of this optimisation approach, and its effectiveness in terms of the quality of the local minima it achieves, are presented in [21]. In practical terms, the conceptual simplicity of the pinning approach, and its ease of algorithmic implementation from

scratch, are clearly desirable. It remains true, however, that more elaborate optimisation schemes [23, 49, 54] may be capable of equivalent or better performance.

To examine this possibility, an alternative multidimensional scaling algorithm using the non-linear least-squares optimisation capability provided by the MATLAB optimisation toolbox was developed. This algorithm, presented as `mds2.m` in Appendix A, is based on the Levenberg-Marquardt optimisation approach [49], and uses the separate residuals function `mdsresiduals.m`. An informal evaluative comparison of the two algorithms across a range of data sets suggested that their performance, in terms of the quality of the solutions they derive, is extremely similar. In most cases, however, the toolbox-based algorithm generated representations significantly more quickly that the pinning-based algorithm when the default of 50 iterations was employed. For this reason, if the MATLAB optimisation toolbox is available, the `mds2.m` algorithm is to be preferred.

In the special case when the Euclidean distance metric is known to be appropriate, it is also possible to use the classical multidimensional scaling algorithm, originally developed in the 1930s [58, 76]. A detailed description of this algorithm is provided by Cox and Cox [18, pp. 22–30], and is implemented by the function `classicalmds.m` presented in Appendix A. Basically, the coordinate locations are generated by performing an eigendecomposition of an inner product matrix derived from the original proximities, with the eigenvectors having the largest associated eigenvalues being returned according to the required number of dimensions. The classical approach to multidimensional scaling has the theoretical advantage of providing a closed form solution, and generates representations more quickly than either of the iterative algorithms. Practical experience suggests that this algorithm can be effective when the proximities tend to satisfy the metric axioms[4] but, in general, derives representations that account for less of the variance than the iterative algorithms. Nevertheless, for proximity data where the objects are known to lie in a Euclidean coordinate space, or for large data sets where it is necessary to sacrifice accuracy for computational speed, the use of the classical algorithm may be appropriate.

3.3 Additive Trees

3.3.1 Generating Additive Trees

The algorithm for generating additive tree representations of proximity data is given by the file `addtree.m` in Appendix B. In general, generating an additive tree from a given proximity matrix requires the determination of a topological tree structure and a set of associated arc lengths. The key observation underlying the algorithm developed here is that, for a given topology, the best-fitting arc lengths are relatively easily determined as

[4]It is often advocated [18] that, when proximity data fails only to satisfy the triangle inequality, it is appropriate to add a constant to each proximity value so that this constraint is met. Although this ensures that the objects can always perfectly be embedded in some Euclidean space, and an analytical solution for finding the smallest applicable constant exists [7], there are grounds to be skeptical of this approach. For proximity data that violates the triangle inequality to the extent that an additive constant is required, the value of this constant is typically so large as dominate the original proximity values. Effectively, this forces the recovered configuration to be embedded in a space of spuriously high dimensionality, since each object is approximately equally distant from each other. Adopting a broader perspective, it seems likely that this sort of proximity data will be better represented by an inherently non-metric approach such as additive clustering.

the solution to a constrained (non-negative) least squares problems. It is the large number of tree structures possible, even for a moderate number of terminal and internal nodes, that makes the generation of additive trees a non-trivial optimisation problem.

Accordingly, the algorithm is founded on a search for an appropriate tree topology using PBIL. The required binary encoding of additive tree topologies is based on the fact that any tree structure is uniquely determined by specifying the parents of each node. That is, for a tree with m internal (non-object) nodes, labelled $i_1, i_2, \ldots, i_m$ and n terminal (leaf) nodes, labelled $t_1, t_2, \ldots, t_n$, listing the parent nodes completely specifies the structure of the tree. There two important restrictions on the choice of parent nodes that guarantee a tree structure. Each parent must be an internal node, and the parent of an internal node i_k must be an internal node i_j where $j < k$. In addition, there are no structural decisions to be made unless an additive tree has two or more internal nodes, and the parent of the second internal node may be assumed to be the root node without loss of generality. Therefore, it is only necessary for the parents of nodes $i_3, i_4, \ldots, i_m, t_1, t_2, \ldots, t_n$ to be represented explicitly. The binary vectors that encode the structure of trees within the PBIL probability vector are simply concatenations of the binary representations of these parents.

It is possible, under this scheme, for inadmissible choices of parent nodes to be represented within the vector. For example, the parent of the fourth internal node, $\mathrm{par}(i_4)$, can be i_1, i_2, or i_3, and therefore demands a two-bit binary representation. The binary strings of 00, 01, and 10 are then taken to correspond to the choice, respectively, of i_1, i_2, or i_3 as the parent node. If, however, the binary string chosen by the stochastic generation process is 11, some other mapping criterion must be employed. Similarly, if there are (say) 3 internal nodes in an additive tree, the parent of each terminal node requires a two-bit representation to span all of the legitimate alternatives, but the bit string 11 does not correspond to an existing internal node. In both of these situations, when invalid parent nodes are implied by the binary representation, the approach adopted here is to over-ride the assignment, and simply set the parent to be the root of the tree. An example of the entire encoding scheme, for a tree containing 6 internal and 4 terminal nodes, is presented below. The 20 element binary representation on the left is broken into internal node and terminal node sections, and then further subdivided into the strings which encode the parent of each of the nodes in the tree.

$$
\begin{array}{l}
10110110110001000010 \mapsto \overbrace{\underbrace{1}_{\mathrm{par}(i_3)=i_2}\ \underbrace{01}_{\mathrm{par}(i_4)=i_2}\ \underbrace{10}_{\mathrm{par}(i_5)=i_3}\ \underbrace{110}_{\mathrm{par}(i_6)=i_1}}^{\text{internal nodes}}\ \overbrace{\underbrace{110}_{\mathrm{par}(t_1)=i_1}\ \underbrace{001}_{\mathrm{par}(t_2)=i_2}\ \underbrace{000}_{\mathrm{par}(t_3)=i_1}\ \underbrace{010}_{\mathrm{par}(t_4)=i_3}}^{\text{terminal nodes}}
\end{array}
$$

For each tree topology represented in this way within the PBIL algorithm, the set of best-fitting arc lengths are found using a non-negative least squares algorithm [36], and the data-fit of the additive tree is evaluated. This evaluation, on which the ordering of the potential PBIL solutions is determined, is again based on the sum squared error for the proximities:

$$\mathrm{SSE} = \sum_{i<j} \left(d_{ij} - \hat{d}_{ij}\right)^2. \tag{17}$$

The additive tree generation algorithm requires as input a square and symmetric proximity matrix and the number of internal nodes to be used in the tree, and returns an adjacency matrix for the nodes specifying the topology of the generated additive tree, a column vector giving the arc length from each node to its parent node, and the variance of the proximity data accounted for by the solution. The algorithm makes default assumptions that can be over-ridden regarding the learning rate λ, the number of evaluative PBIL trials to be made without improvement before terminating, the size of the set of potential solutions within PBIL, the number of solutions in this set to be used in competitive learning, and the probability and shift values associated with mutation.

A transcript of a sample MATLAB session, using the additive tree algorithm to recover the representation shown in Figure 2 from the given proximity matrix, is shown below. It is worth noting that, because the algorithm can take some time to terminate, messages are periodically displayed during its operation, reporting the attainment of an improved solution, or the passing of a block of 50 evaluations without improvement.

```
>> help addtree

  ADDTREE additive tree (michael.d.lee@dsto.defence.gov.au)
  [adjacency,lengths,vaf]=addtree(distance,nodes,learnrate,maxtrials,batchsize,batchlearn,mutprob,mutshift)

  DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
  NODES specifies the number of internal nodes to place in the tree (required)
  LEARNRATE specifies the learning rate used in optimisation (default=0.1)
  MAXTRIALS specifies the number of trees evaluated without improvement before terminating (default=3000)
  BATCHSIZE specifies the size of the batch of potential tree solutions (default=50)
  BATCHLEARN specifies the number of best solutions in the batch used in learning (default=1)
  MUTPROB specifies the probability of 'mutating' the PBIL probability vector (default=0.02)
  MUTSHIFT specifies the extent of a 'mutation' shift (default=0.05)

  ADJACENCY returns an (N+NODES)x(N+NODES) adjacency matrix defining the tree topology
  LENGTHS returns a vector of length (N+NODES) containing the arc-lengths for the tree
  VAF returns the variance of the distance values accounted for by the solution

>> d=[00 15 20 18 20;
      15 00 25 23 25;
      20 25 00 06 20;
      18 23 06 00 18;
      20 25 20 18 00];

>> [adjacency,lengths,vaf]=addtree(d,3,.1,300)

better tree found: accounts for 41.73 percent of the variance
better tree found: accounts for 41.73 percent of the variance
better tree found: accounts for 76.98 percent of the variance
better tree found: accounts for 84.01 percent of the variance
better tree found: accounts for 84.01 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
better tree found: accounts for 100.00 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
150 trials have elapsed without improvement
200 trials have elapsed without improvement
250 trials have elapsed without improvement
300 trials have elapsed without improvement
adjacency =
     0     1     1     0     0     0     0     1
     1     0     0     1     1     0     0     0
     1     0     0     0     0     1     1     0
     0     1     0     0     0     0     0     0
```

```
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     1     0     0     0     0     0
     1     0     0     0     0     0     0     0
lengths =
         0
    5.0000
    6.0000
    5.0000
   10.0000
    4.0000
    2.0000
   10.0000
vaf =
     1

>>
```

3.3.2 Displaying Additive Trees

The algorithm for displaying additive trees, listed in the file `displaytree.m` in Appendix C, is based on the novel[5] approach presented in [66]. Rather than using a traditional display, of the type shown in Figure 2, this new display is based on a two-dimensional spatial layout of the internal and terminal nodes that meets the following criteria: The Euclidean distance between any pair of nodes that is connected in the tree structure must approximate the associated arc length of the additive tree, and the terminal nodes should be positioned so that their distances approximate the original proximity values, as in multidimensional scaling. The algorithm developed here used the pinning variant of gradient descent based optimisation to find such a configuration, and then uses a given set of labels to display the additive tree.

Formally, each pair of nodes that are connected within the tree topology are considered to form a set $\mathbf{C}$, while all of the terminal node pairs corresponding to stimulus pairs form a set $\mathbf{T}$. Accordingly, the iterative learning rule is based on the derivative of a sum squared error measure that is comprised of two parts:

$$\mathrm{SSE}\left(\mathbf{x}_1,\ldots,\mathbf{x}_n;\mathbf{y}_1,\ldots,\mathbf{y}_m\right) = \alpha \sum_{(i,j)\in\mathbf{C}} \left(d_{ij} - \hat{d}_{ij}\left(\mathbf{x}_i,\mathbf{x}_j\right)\right)^2 + \sum_{(i,j)\in\mathbf{T}} \left(d_{ij} - \hat{d}_{ij}\left(\mathbf{y}_i,\mathbf{y}_j\right)\right)^2, \tag{18}$$

where $\mathbf{x}_1,\ldots,\mathbf{x}_n \in \mathbf{R}^2$ are the coordinate locations of the terminal nodes, $\mathbf{y}_1,\ldots,\mathbf{y}_m \in \mathbf{R}^2$ are the coordinate locations of the internal nodes, and $\alpha > 1$ is a weighting parameter used to give relatively greater emphasis to the preservation of arc lengths.

The algorithm requires as input the proximity matrix for which the additive tree representation was derived, the adjacency matrix that defines the tree, the arc lengths associated with the adjacency matrix[6], and a set of labels for the objects being represented. Default values are assigned to the number of characters in each label to be included in the display (with a legend automatically being drawn if the entire label is not used), the size of the

[5] but almost entirely undocumented

[6] Both the adjacency matrix and length vector assume the same form as that provided by the `addtree.m` algorithm.

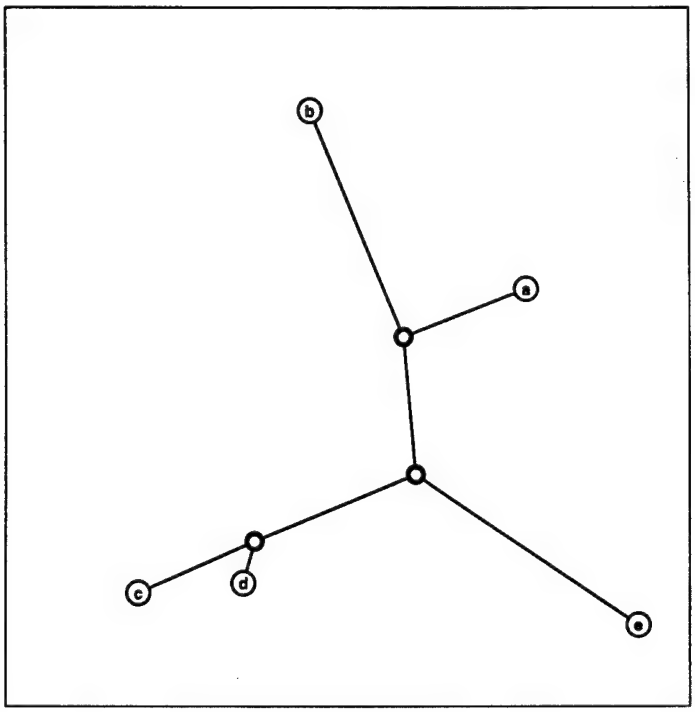

Figure 4: The figure generated by the additive tree display algorithm.

arc preservation weight α, the number of optimisation iterations to be performed, and the learning rate associated with this optimisation. As an option, the algorithm is able to return the coordinates of the points at which the nodes are drawn, and the correlation coefficient between the arc lengths specified within the tree and the lengths of the arcs actually drawn in the display.

The following is the transcript of a sample MATLAB session using the additive tree display algorithm to draw the additive tree from Figure 2, as previously recovered using the algorithm for generating additive trees. The actual display generated as a MATLAB figure is shown in Figure 4:

```
>> help displaytree

  DISPLAYTREE displays an additive tree (michael.d.lee@dsto.defence.gov.au)
  [points,corr]=displaytree(distance,adjacency,lengths,labels,characters,preserveweight,iterations,learnrate)

  DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
  ADJACENCY is an (N+NODES)x(N+NODES) adjacency matrix defining the tree topology, as returned by addtree
  LENGTHS returns a vector of length (N+NODES) containing the arc-lengths for the tree, as returned by addtree
  LABELS is a string array of containing N labels for the terminal nodes (required)
  CHARACTERS specifies how many characters to include in the display (default=0 displays the entire label)
  PRESERVEWEIGHT specifies the relative emphasis given to preserving lengths in the tree (default=10)
  ITERATIONS specifies the number of optimisation iterations performed (default=50)
  LEARNRATE specifies the learning rate used in optimisation (default=0.05)

  POINTS returns the coordinate locations of each of the internal and terminal nodes (optional)
  CORR returns the correlation between the specified lengths and the displayed lengths (optional)
```

```
>> labels=char('a','b','c','d','e');

>> [points corr]=displaytree(d,adjacency,lengths,labels,0,1,100,0.1)
points =
         0         0
   -0.5008    6.4969
   -6.4977   -3.1484
    4.4220    8.7806
   -4.2600   17.1614
  -11.1731   -5.5798
   -6.9492   -5.1133
    8.9440   -7.1131
corr =
    0.9894

>>
```

3.4 Additive Clustering

The algorithm for generating additive clustering representations of similarity data is given by the file `adclus.m` in Appendix D. The basic approach is similar to that used to generate additive trees, and relies on the observation that the determination of optimal cluster weights is straightforward for a given pattern of assignment of objects to clusters. It is the flexibility with which these assignments can be made, in the sense that there are none of the constraints of partitioning or hierarchical clustering, that renders additive clustering a difficult combinatorial optimisation problem.

Accordingly, the algorithm concentrates on searching for an appropriate pattern of cluster membership using PBIL. Formally, when deriving an additive clustering representation involving n objects and m clusters, this pattern is defined by an $n \times m$ matrix of binary membership variables $\mathbf{F} = [f_{ik}]$, where:

$$f_{ik} = \begin{cases} 1 & \text{if object } i \text{ is in cluster } k \\ 0 & \text{otherwise} \end{cases}.$$

Given this information, the cluster weights w_k may be found by again solving a non-negative least squares problem [36] in relation to the sum squared error of the similarities:

$$\text{SSE} = \sum_{i<j} (s_{ij} - \hat{s}_{ij})^2 , \tag{19}$$

where the modelled similarities $\hat{s}_{ij}$ are calculated as the sum of the weights of the clusters to which both the ith and jth objects belong, as follows:

$$\hat{s}_{ij} = \sum_k w_k f_{ik} f_{jk}. \tag{20}$$

In much the same way as the algorithm for generating additive trees, the additive clustering algorithm uses the sum squared error as an evaluative measure with which to modify the PBIL probability vector. The algorithm requires as input a square and

symmetric similarity matrix and the number of clusters to be used, and returns the cluster assignment matrix, a column vector giving the weights of the clusters, and the variance of the similarity data accounted for by the solution. As before, the algorithm makes default assumptions regarding the learning rate λ, the number of evaluative PBIL trials to be made without improvement before terminating, the size of the set of potential solutions within PBIL, the number of solutions in this set to be used in competitive learning, and the probability and shift values associated with mutation.

Because of some technical issues that arise when using the VAF measure with additive clustering [47, 48], the algorithm augments the universal cluster, including all objects, to each potential solution. The weight of this cluster appends an additive constant to the similarity model given in Equation 20, and ensures that the VAF measure is valid.

The following is the transcript of a sample MATLAB session using the additive clustering algorithm to recover the representation shown in Figure 3 from the given similarity matrix. Once again, a number of messages are periodically displayed during the operation of the algorithm. It is also worth noting that, because of the automatic inclusion of the universal cluster presented in Figure 3, only the two additional clusters are requested:

```
>> help adclus

  ADCLUS additive clustering (michael.d.lee@dsto.defence.gov.au)
  [clusters,weights,vaf]=adclus(similarity,numberclusters,learnrate,maxtrials,batchsize,batchlearn,mutprob,mutshift)

  SIMILARITY is an NxN symmetric matrix of pairwise similarities (required)
  NUMBERCLUSTERS specifies the number of clusters to use (required)
  LEARNRATE specifies the learning rate used in optimisation (default=0.1)
  MAXTRIALS specifies the number of trees evaluated without improvement before terminating (default=3000)
  BATCHSIZE specifies the size of the batch of potential tree solutions (default=50)
  BATCHLEARN specifies the number of best solutions in the batch used in learning (default=1)
  MUTPROB specifies the probability of 'mutating' the PBIL probability vector (default=0.02)
  MUTSHIFT specifies the extent of a 'mutation' shift (default=0.05)

  CLUSTERS returns an Nx(NUMBERCLUSTERS+1) matrix defining derived cluster membership plus the universal cluster
  WEIGHTS returns a vector of length (NUMBERCLUSTERS+1) containing the weights of the clusters
  VAF returns the variance of the similarity values accounted for by the solution

>> s=[00 17 17 05 05;
      17 00 17 05 05;
      17 17 00 14 05;
      05 05 14 00 05;
      05 05 05 05 00];

>> [clusters,weights,vaf]=adclus(s,2,.1,200,20,1)
better clustering found: accounts for 20.13 percent of the variance
better clustering found: accounts for 20.13 percent of the variance
better clustering found: accounts for 45.29 percent of the variance
better clustering found: accounts for 77.64 percent of the variance
better clustering found: accounts for 78.66 percent of the variance
better clustering found: accounts for 82.61 percent of the variance
50 trials have elapsed without improvement
better clustering found: accounts for 82.61 percent of the variance
better clustering found: accounts for 83.99 percent of the variance
better clustering found: accounts for 83.99 percent of the variance
50 trials have elapsed without improvement
better clustering found: accounts for 100.00 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
150 trials have elapsed without improvement
200 trials have elapsed without improvement
clusters =
```

```
     0     1     1
     0     1     1
     1     1     1
     1     0     1
     0     0     1
weights =
    9.0000
   12.0000
    5.0000
vaf =
     1

>>
```

4 Complexity Analysis

4.1 Ockham's Razor

The accuracy of a derived multidimensional scaling, additive tree or additive clustering representation may be measured according to its success in modelling the similarity or proximity values from which it was generated. In particular, the SSE measures that the algorithms seek to minimise, or the VAF measures they return, both provide an indication of the data-fit associated with a representation. It should be clear, however, that these indicators can often be improved by including additional dimensions for multidimensional scaling representation, or by including additional internal nodes for additive tree representations. It is also well known [70] that, through the inclusion of enough clusters, additive clustering representations can always be made to fit any similarity matrix without error.

These considerations raise the issue of complexity in relation to the representational models being generated. Under the pervasive modelling stricture variously referred to as 'the principle of parsimony' or 'Ockham's Razor'[7], models should only be made as complicated as the improvement in data-fit resulting from additional complexity warrants. This means that representations should be developed by explicitly considering the trade-off between accommodating the data they seek to explain, and minimizing their complexity. From a general model-theoretic perspective, finding an appropriate balance between these competing demands facilitates the fundamental goals of achieving substantive interpretability, explanatory insight, and the ability to generalize accurately.

In terms of the representational models being considered here, the issue of complexity control is naturally interpreted in parametric terms. Most generally, there are at least two identifiable complexity components for any given model, relating to the number of parameters that are used, and the functional form of their interaction [50]. Intuitively, the primary determinant of model complexity for multidimensional scaling, additive tree, and additive clustering representations would seem to be, respectively, the number of dimensions, internal nodes, and clusters used. These are essentially measures of the 'number of parameters' component of model complexity, since the addition of dimensions, nodes and

[7]Ockham's Razor is named after British medieval philosopher William of Ockham (1285–1349), on the basis of the observation "entia non sunt multiplicanda sine necessitate" ("entities should not be multiplied beyond necessity"). According to [29], "although it is not clear that he actually used these words ... he certainly said things very like that".

clusters adds extra degrees of representational freedom primarily by introducing more free parameters into the model fitting process.

The functional form component of model complexity is more subtle, and relates to the ways in which the parameters may interact. For multidimensional scaling representations, this is determined by the form of the distance metric used within a coordinate space of any given dimensionality. For additive trees, it relates to the different tree topologies that might be constructed using the same number of internal and external nodes. For additive clustering models, it relates to the various patterns of encompassment and overlap that a fixed number of clusters might assume. In general, the complexity effects of these changes are poorly understood, although there is a suggestion [65, 38] that the Euclidean metric is less complicated than many others, and some attempts at quantifying cluster structure complexity for additive cluster models are presented in [40]. What evidence is available, however, suggests that this functional form complexity component has significantly less impact on overall model complexity than the simple, and readily quantified, number of parameters component.

4.2 Bayesian Information Criteria

Accordingly, one way of quantitatively addressing the complexity issue is through the Bayesian Information Criterion (BIC), which is based on a parametric measure of model complexity [59, 33, 50]. The BIC takes the general form:

$$\mathrm{BIC} = -2\log p(\mathrm{ML}) + P\log N,$$

where $p(\mathrm{ML})$ is the maximum likelihood estimate of the model, P is the number of parameters in the model, and N is the sample size. Basically, the first term of the BIC relates to the data-fit properties of a model, while the second term penalises the model according to the number of parameters it uses to achieve this fit. Qualitatively, it can be seen that this measure increases whenever either the number of parameters increases, or when the accommodation of the data worsens. Accordingly, the candidate model with the minimal BIC value is to be preferred[8].

One of the most desirable properties of the BIC measure is that, unlike alternative regularisation-based approaches to controlling model complexity [28], the relative weight given to the data-fit and complexity components is not free to vary in arbitrary ways. As argued by Kass and Raftery [33], the probabilistic basis of the BIC provides a meaningful interpretative scale defined by odds ratios, and removes the need for the consensual subjective calibration[9] of data-fit and complexity, and lies on the same scale as likelihood ratio test statistics. Indeed, it is well known that the difference between BIC values for two competing models approximates twice the logarithm of the so-called 'Bayes Factor',

[8]It is worth noting that the BIC, which is derived as an asymptotic approximation to the Bayesian posterior probability of a model under the assumption of uniform priors [59], is closely related [77] to well-known minimum description length (MDL) evaluative measures [56], through the connection provided by Shannon's Optimal Coding Theorem [60].

[9]'consensual subjective calibration' is a euphemism for 'fudge factor'

which is the ratio of the posterior to prior odds for the two models, and effectively quantifies the evidence for the various models provided by the data at hand. Using the BIC to compare competing models means that the only 'subjective' decision that remains to be made is the natural and unavoidable one relating to the standards of scientific evidence being applied.

A maximum likelihood estimate for multidimensional scaling, additive tree, and additive clustering models may be formulated [72] by assuming that each similarity or proximity value in the matrix $\mathbf{X} = [x_{ij}]$ has a Gaussian distribution with common variance σ^2. In this case, the estimates of these similarities or proximities, $\hat{\mathbf{X}} = [\hat{x}_{ij}]$, associated with a derived representation containing P parameters, is assigned a likelihood:

$$p\left(\mathbf{X} \mid P, \hat{\mathbf{X}}\right) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i<j}\left(x_{ij}-\hat{x}_{ij}\right)^2\right). \tag{21}$$

It is worth noting that twice the negative logarithm of this likelihood, as used in the data-fit component of the BIC, is given by:

$$-2\log p\left(\mathbf{X} \mid P, \hat{\mathbf{X}}\right) = \frac{1}{\sigma^2}\sum_{i<j}\left(x_{ij}-\hat{x}_{ij}\right)^2, \tag{22}$$

which is simply the SSE measure, as scaled by the precision variance of the constraining data under the Gaussian distribution model. This term of the BIC is the same for multidimensional scaling, additive tree, and additive clustering models. The sample size N does not vary either, since all of the representations are generated from an $n \times n$ symmetric matrix of pairwise similarities or proximities, containing $n\left(n-1\right)/2$ unique object pairings.

The number of parameters, P, however, differs across the three representational models. For a multidimensional scaling representation, the inclusion of each additional dimension results in the need to determine an extra parameter, in the form of a coordinate value for the new dimension, for each object being represented. This means that a multidimensional scaling representation of n objects using m dimensions contains a total of mn parameters. Accordingly, the BIC for a multidimensional scaling model takes the form:

$$\mathrm{BIC}_{\mathsf{mds}} = \frac{1}{\sigma^2}\sum_{i<j}\left(d_{ij}-\hat{d}_{ij}\right)^2 + mn\log\left(\frac{n(n-1)}{2}\right). \tag{23}$$

For an additive tree representation, once the first internal node has been established, each additional internal node introduces one new arc length parameter to the model fitting process. There are also arc length parameters associated with each of the terminal nodes, meaning that an additive tree for n objects incorporating m internal nodes has a total of $(m+n-1)$ parameters. This means that the BIC for an additive tree model is given by:

$$\mathrm{BIC}_{\mathsf{addtree}} = \frac{1}{\sigma^2}\sum_{i<j}\left(d_{ij}-\hat{d}_{ij}\right)^2 + (m+n-1)\log\left(\frac{n(n-1)}{2}\right). \tag{24}$$

For an additive clustering model, each of the clusters requires the determination of a parameter, in the form of a weight. Therefore, the total number of parameters in an additive clustering representation using m clusters for n objects is simply m, and the BIC takes the form:

$$\mathrm{BIC}_{\mathsf{adclus}} = \frac{1}{\sigma^2}\sum_{i<j}(s_{ij} - \hat{s}_{ij})^2 + m\log\left(\frac{n(n-1)}{2}\right). \tag{25}$$

The evaluation of these BIC measures can be accomplished using the function `bic.m`, listed in Appendix E. This function requires as input a string indicating whether a multidimensional scaling, additive tree, or additive clustering is involved, the similarity or proximity data from which the representation was generated, the variance of the data accounted for by the representation, the assumed level of data precision, and the number of dimensions, internal nodes, or clusters (not including the universal cluster) used. The following is a transcript of a sample MATLAB session where the similarity matrix in Figure 3 is supplied to `adclus.m` to generate additive clustering models with 1, 2 and 3 clusters. The `bic.m` function is then used to calculate BIC values for each of the three models, given an assumed level of data precision $\sigma = 0.1$. It can be seen that the model shown in Figure 3, with 2 clusters and the universal cluster, corresponds to the minimal BIC value.

```
>> s=[00 17 17 05 05;
      17 00 17 05 05;
      17 17 00 14 05;
      05 05 14 00 05;
      05 05 05 05 00];

>> [clusters1,weights1,vaf1]=adclus(s,1,.1,200,20,1);
better clustering found: accounts for 20.13 percent of the variance
better clustering found: accounts for 77.64 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
150 trials have elapsed without improvement
200 trials have elapsed without improvement

>> [clusters2,weights2,vaf2]=adclus(s,2,.1,200,20,1);
better clustering found: accounts for 43.48 percent of the variance
better clustering found: accounts for 77.64 percent of the variance
better clustering found: accounts for 83.99 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
150 trials have elapsed without improvement
better clustering found: accounts for 83.99 percent of the variance
better clustering found: accounts for 100.00 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
150 trials have elapsed without improvement
200 trials have elapsed without improvement

>> [clusters3,weights3,vaf3]=adclus(s,3,.1,200,20,1);
better clustering found: accounts for 10.56 percent of the variance
better clustering found: accounts for 30.43 percent of the variance
better clustering found: accounts for 38.16 percent of the variance
better clustering found: accounts for 77.64 percent of the variance
better clustering found: accounts for 83.99 percent of the variance
50 trials have elapsed without improvement
better clustering found: accounts for 100.00 percent of the variance
50 trials have elapsed without improvement
100 trials have elapsed without improvement
```

```
150 trials have elapsed without improvement
200 trials have elapsed without improvement

>>  [vaf1 vaf2 vaf3]
ans =
    0.7764    1.0000    1.0000

>> help bic

  BIC Bayesian information criterion (michael.d.lee@dsto.defence.gov.au)
  value=bic(bictype,data,vaf,sigma,dims_nodes_clusters)

  BICTYPE specifies the model type using the string 'mds', 'addtree', or 'adclus' (required)
  DATA is the NxN symmetric matrix of similarities or proximities (required)
  VAF is variance of the similarity values accounted for by the representation (required)
  SIGMA is the assumed level of data precision (required)
  DIMS_NODES_CLUSTERS gives the number of dimensions, internal nodes, or clusters in the representation (required)

  VALUE returns the Bayesian information criterion value

>> bic1=bic('adclus',s,vaf1,.1,1);

>> bic2=bic('adclus',s,vaf2,.1,2);

>> bic3=bic('adclus',s,vaf3,.1,3);

>> [bic1 bic2 bic3]
ans =
  1.0e+003 *
    6.9475    0.0069    0.0092

>>
```

In this general way, a number of competing multidimensional scaling, additive tree, or additive clustering representations with different levels of data-fit and parametric complexity, may be compared in terms of their BIC values. Clearly, however, the outcome of these comparisons will depend upon assumptions made regarding the variance of the similarity or proximity data, as formalized by specifying a value for σ.

4.3 Measuring Data Precision

In effect, σ quantifies the inherent precision of a matrix of similarity or proximity data, and provides an indication of the level of data-fit with which multidimensional scaling, additive tree or additive clustering representations should seek to model the stated relationships between objects. If the given similarity or proximity values are known to be very accurate, for example, the inclusion of additional spatial dimensions, internal nodes, or featural clusters may well be warranted to capture all of the detail provided. If, however, the constraining data are imprecise, then it may be appropriate to model only the major representational trends using a relatively simple model. An important property of this conception is that precision is a property of a similarity or proximity matrix itself, and is independent of its use within any representational framework. This means that σ should be derived from an understanding of the process by which the similarity or proximity values themselves were generated, and not estimated as a parameter within the process of fitting a particular representational model.

One means of determining data precision, particularly applicable within the common

experimental situation where the final similarity or proximity matrix is derived by averaging across the ratings provided by a number of subjects, is to calculate σ as the average of the standard deviations for each of the pooled cells in the matrix. In an identification experiment, a signal detection type of analysis of the obtained confusion matrix may be used to furnish a useful estimate of data precision. For similarity matrices generated from objects with list or properties, information regarding the distribution of these properties could be incorporated into an estimation procedure. In general, there are a number of plausible means by which a σ value, or range of σ values, may be determined for similarity or proximity matrices generated by a variety of means. The key question is always one of the accuracy of the similarity or proximity entries in the matrix, since σ quantifies the variance of the distribution from which each of these values is assumed to be drawn.

As a concrete example of one of these approaches, consider an experiment [34] examining the ability of a particular cognitive model to emulate human performance on various categorisation tasks. The experiment involved a set of 16 geometric stimuli and, in part, required a total of 400 subjects to rate the similarity of each possible pair of these stimuli on a 9-point scale. A similarity matrix for the stimulus set was then formed by averaging the individual similarity matrices of each of the subjects, and then normalising. Although not considered in the original study [34], a separate analysis [38] of these individual matrices found that the standard deviations for the cells in the final matrix ranged between 0.065 and 0.166, with a mean value of 0.125.

Equivalent analyses of this type have been conducted on other similarity and proximity data that were also constructed by averaging the individual ratings provided by subjects. Results of these analyses provide σ values ranging from a minimum of 0.129 to a maximum of 0.283, with a mean of 0.203 for simple geometric stimuli called 'Shepard circles' [4], and means of 0.183, 0.082, 0.148, and 0.127 for four individual subjects, formed by averaging their repeated ratings of 12 deformed circles across four blocks, using a sliding continuous scale [41]. Effectively, these values quantify the agreement across different subjects, or across the same subject on different occasions, in their ratings of the stimulus pairs. Close agreement, which provides precise data, is heralded by small values of σ, whereas noisy or imprecise data are indicated by large values of σ.

Unfortunately, when the raw data needed to form estimates of precision in this, or any other rigorous sort of way are not available, the choice of σ must be made on more subjective and heuristic grounds. In this case, the results cited above might provide some broad guidance. For normalised similarity or proximity matrices, a σ value of about 0.10 seems to correspond to reasonably precise data, while values over 0.20 seem to correspond to particularly imprecise data. It is possible to consider the implications for model complexity across a plausible range of data precision by examining the pattern of change of the BIC for each of a number of specified σ values[10]. In this way, the sensitivity of a

[10]The nature of the approximations used to derive Equation 22, however, means that it is not valid to compare BIC values across different assumed levels of precision. Each BIC measure is accurate only up to a factor that is constant only for a fixed value of σ. This approximation is appropriate, given that, although it seems tempting at first, the joint minimisation of the BIC in terms of the number of parameters and data precision does not make sense. Since σ is a fixed, although possibly unknown value, dependent upon the way in which similarity or proximity data are obtained, it is not free to vary once the task of representational modelling has commenced. In addition, knowing what combination of data precision and parameter numbers minimises the BIC for a given data set does not have implications for subsequent data collection. The aim should always be to provide similarity or proximity data that are as precise as possible.

conclusion regarding the appropriate complexity to the true, but unknown, precision of the similarity or proximity data may be assessed.

Despite the generality of these sorts of guidelines, the role of σ, in forcing an explicit and quantitative assumption to be made about data precision before fitting a representational model, is an important one. It is possible for two similarity or proximity matrices to be identical in terms of their individual entries, but to have different associated levels of precision. Under the approach being advocated here, these two matrices are likely to demand representational models with different levels of complexity. This allows precise data collected, say, from domain experts exhibiting close agreement in their judgments, to be fit by a detailed model with many parameters, while ensuring that less precise data are not over-fit by a similarly complex model.

5 Illustrative Examples

5.1 Multidimensional Scaling Representation of Colours

As an example of the multidimensional scaling model of representation, consider similarity data reported by Ekman [25, Table 1]. This data was constructed by pooling ratings made on a 5-point scale by 31 subjects for 14 colours, specified by their wavelengths. After converting the similarity measures to proximities using a logarithmic transformation, multidimensional scaling representations were derived[11] in spaces of 1-7 dimensions operating under the Euclidean distance metric using `mds.m`.

Unfortunately, the individual ratings provided by subjects are not available, meaning that some reasonable assumption must be made regarding the precision of the data. On the basis of the various σ values detailed earlier, it could be argued that the precision of the colour data lies somewhere in the broad range between 0.10 and 0.20. Accordingly, Figure 5 shows the VAF goodness-of-fit measure, and the pattern of change of the BIC measures across the 7 dimensionalities using σ values of 0.10, 0.15 and 0.20. In the case of reasonably precise data, it can be seen that the BIC measure is minimal at the value of 2, suggesting that a two dimensional spatial representation is appropriate. This conclusion concurs with both an established theoretical understanding of the colours being considered, and previous multidimensional scaling studies examining this data [37, 62].

The multidimensional scaling representation itself is depicted in Figure 6, and takes the form of the colour 'circle' or 'horse-shoe' [67] of the type originally anticipated by Newton [51, 66]. Basically, the uni-dimensional wavelength continuum is bent so that the two extremes are near each other, in accordance with the high degree of psychological similarity between red and violet colours evident in the ratings data. This representation provides a clear example of the need, whatever representational model is employed, to use similarity or proximity data appropriate to the task at hand. The colour representation given in Figure 6 is likely to be well suited in a context involving human observer, such

[11] As with all of the algorithms presented here, the multidimensional scaling algorithm is not guaranteed to find a globally optimal representation. Accordingly, it is prudent to make several attempts at generating any particular representation, and choose the best fitting one. Experience suggests that this is particularly necessary when dealing with non-Euclidean distance metrics.

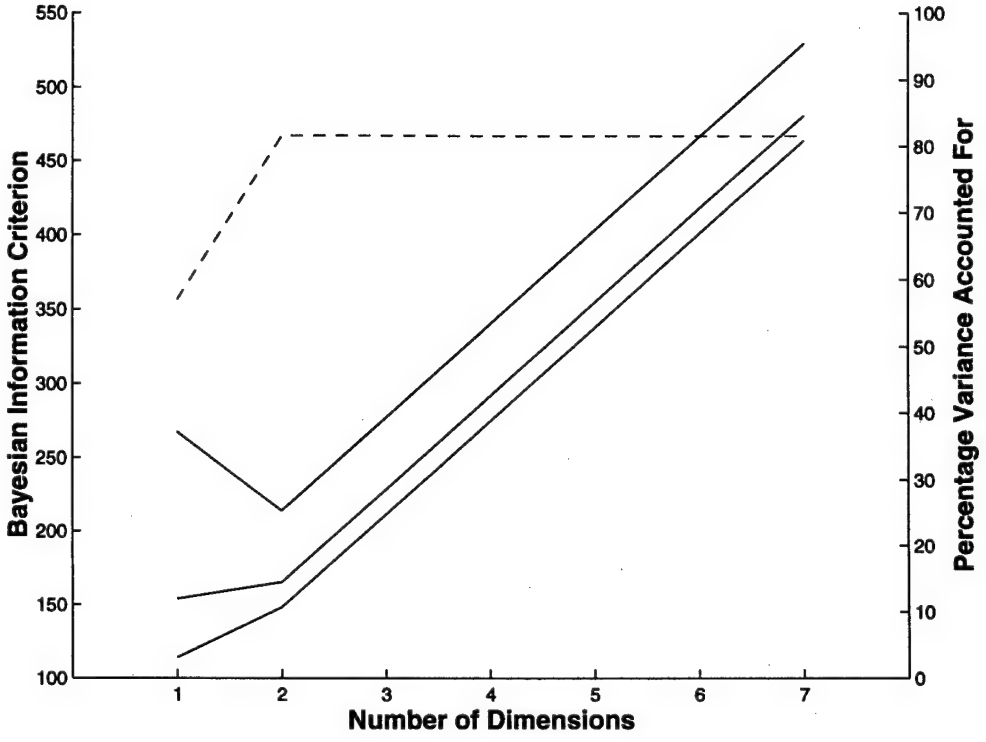

Figure 5: BIC values (left hand scale) for the colour data, using σ values of 0.10, 0.15, and 0.20 (top to bottom), across representational spaces with 1 through 7 dimensions. The variance account for (right hand scale) by the best fitting configuration of each dimensionality is shown by the broken line.

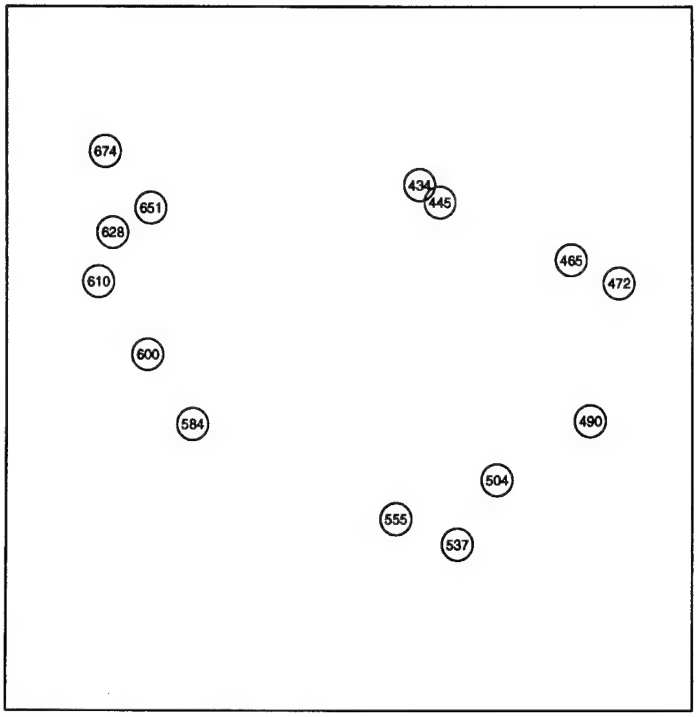

Figure 6: Multidimensional scaling representation of the 14 colours, as labelled by their wavelengths in nm.

as the visual detection of objects in a display. It is less likely to be useful in the context of understanding colour phenomena fundamentally rooted in the physical sciences, where the conventional wavelength spectrum might be preferred. In general, a multidimensional scaling, additive tree, or additive clustering representation can only be as useful as the similarity or proximity data from which it is generated, and it is worth making some effort to provide relevant data.

5.2 Additive Tree Representation of Risks

As an example of constructing an additive tree representation, consider data measuring the similarity of 18 different 'risks', as reported by Johnson and Tversky [32, Table A1, lower triangular half]. These data were obtained by pooling the ratings of pairwise similarity made by a number of subjects on a 9-point scale. After converting the similarities to proximities using a linear transformation, additive trees with between 2 and 16 internal nodes were derived using `addtree.m`.

Once again, the ratings for the individual subjects are not available, although a reliability measure is provided [32], giving a correlation of 0.90 between two large randomly selected samples of subjects. While this measure is not prescriptive in terms of specifying σ, some Monte Carlo exploration [39] has shown that it is consistent with adopting the

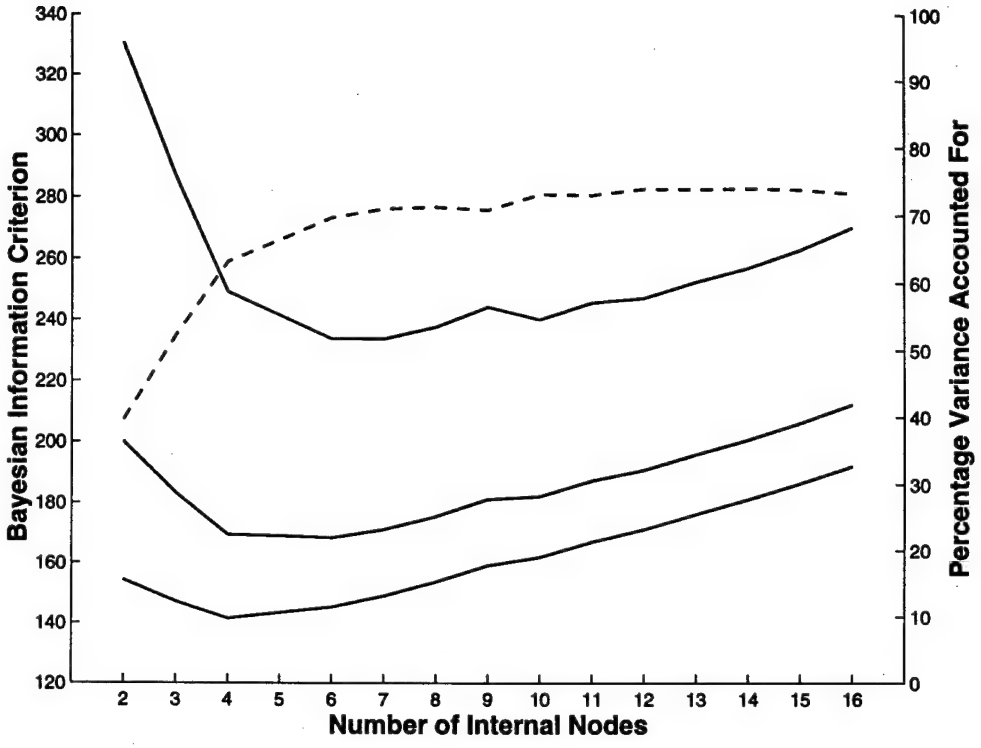

Figure 7: Bayesian Information Criterion (left hand scale, solid line) for σ values of 0.10, 0.15 and 0.20 (top to bottom), and Percentage of Variance Accounted For (right hand scale, broken line) values for the risk data, using additive trees with between 2 and 16 internal nodes.

same range of σ values as was used for the colour data.

The results of the consequent BIC analysis, together with the VAF measures for the best-fitting tree with each number of internal nodes, are shown in Figure 7. At the $\sigma = 0.10$ level, the pattern of change of the BIC suggests using an additive tree with 6 or 7 internal nodes. As the assumed precision decreases, however, the use of only 4 or 5 internal nodes is indicated. While these sorts of conclusion are necessarily general, they do provide a basis on which the number of internal nodes that should be included in an additive tree model may be restricted to a manageable range.

Using the display algorithm `displaytree.m`, Figure 8 shows the best-fitting additive tree for the risk data containing 5 internal nodes. A weighting parameter value of $\alpha = 100$ was used, and the arc lengths actually displayed were observed to have a correlation greater than 0.99 with the arc lengths specified by the model. The internal nodes correspond to clusters of risks that each seem amenable to meaningful subjective interpretation as (clockwise from top) 'natural disasters', 'technological disasters', 'violent acts', 'illnesses' and 'accidents'. The fact that the generation algorithm, coupled with the complexity analysis, derived additive tree models that aligned internal nodes with meaningful distinctions is particularly encouraging. It is precisely this sort of substantive interpretation, which

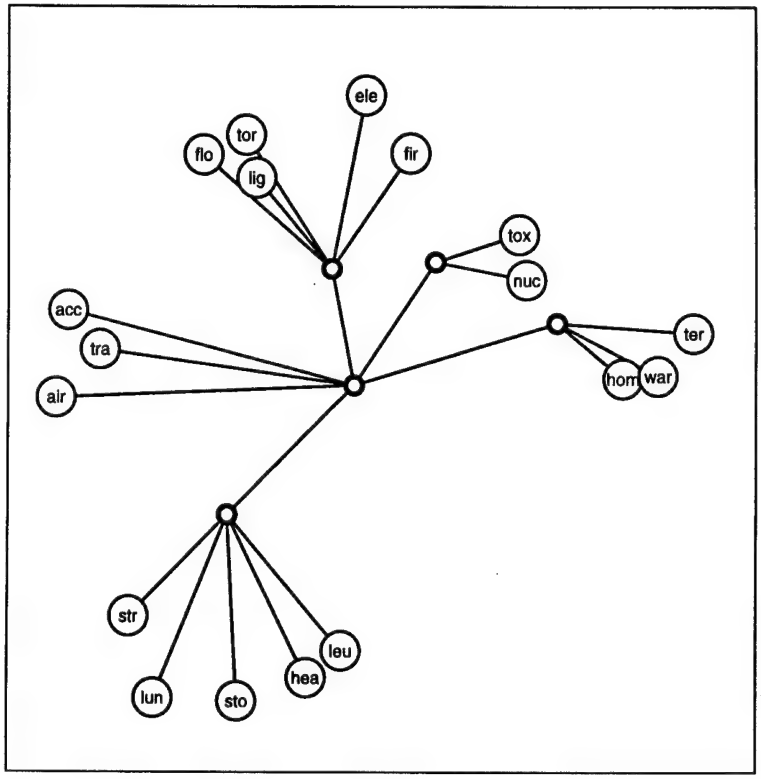

Figure 8: Best-fitting additive tree with 5 internal nodes for the risk data.

primarily results from additive tree models not being over-fit to proximity data, that the current approach should facilitate.

5.3 Additive Clustering Representation of Numerals

As an example of the additive clustering representational approach, consider the similarity matrix relating to the 10 Arabic numerals '0', '1',...,'9' generated by pooling judgments of their 'abstract conceptual' similarity across three conditions of stimulus presentation, presented by Shepard, Kilpatrick and Cunningham [71]. Using the `adclus.m` algorithm, additive clustering representations with between 6 and 12 clusters were generated. Once again, BIC measures corresponding to σ values of 0.10, 0.15, and 0.20 were calculated, and they are shown together with the VAF measure in Figure 9.

The pattern of change of the BIC suggests the use of up to 10 clusters is appropriate if the data are assumed to be precise, and the restriction to as few as 6 clusters when imprecision is assumed. As was the case in the additive tree example, this is a general conclusion that constrains the number of clusters the additive clustering representation should use to a manageable range.

The 8 cluster representation that is favoured for the moderate level of data precision

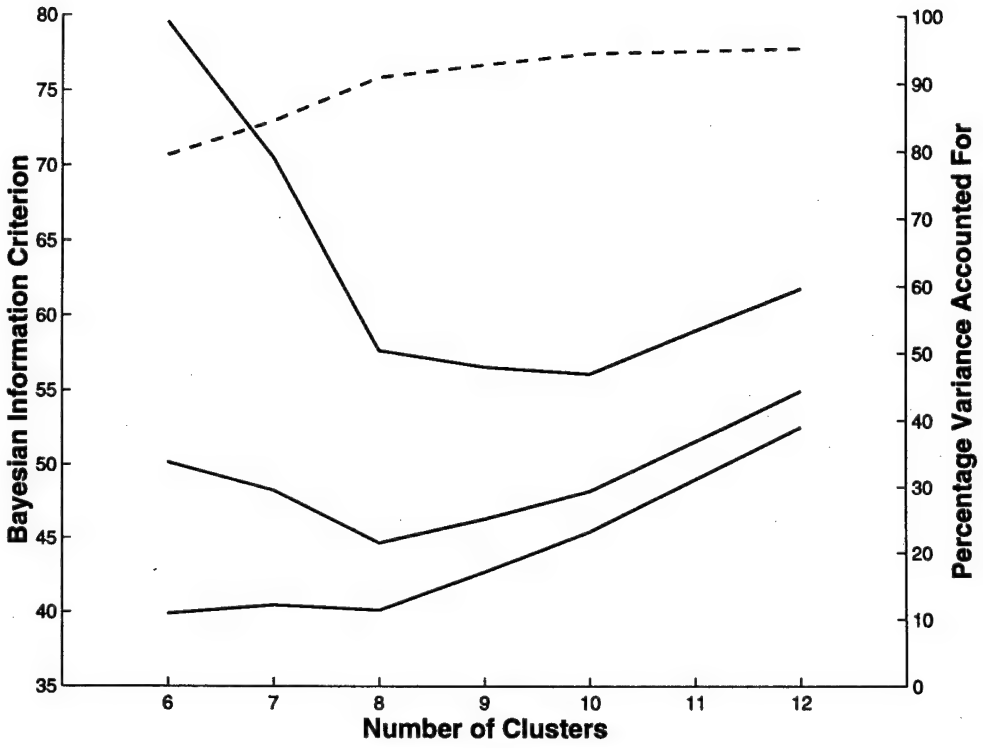

Figure 9: Bayesian Information Criterion (left hand scale, solid line) for σ values of 0.10, 0.15 and 0.20 (top to bottom), and Percentage of Variance Explained (right hand scale, broken line) values for the Arabic numeral data, using additive clustering models with between 6 and 12 clusters.

Table 1: The 8 cluster Arabic numeral representation.

Stimuli In Cluster									Weight
		2		4				8	0.444
0	1	2							0.345
			3			6		9	0.331
						6	7	8	9	0.291
		2	3	4	5	6				0.255
	1		3		5		7		9	0.216
	1	2	3	4						0.214
				4	5	6	7	8		0.172
additive constant										0.148

$\sigma = 0.15$ is detailed in Table 1. It can be seen that this representation incorporates clusters relating to both the numerical magnitude (e.g., $\{1,2,3,4\}$ and $\{6,7,8,9\}$) of the numbers, and various arithmetic concepts (e.g., $\{2,4,8\}$, $\{3,6,9\}$ and $\{1,3,5,7,9\}$). The usefulness of the representational flexibility afforded by additive clustering is particularly evident in this regard, since the ability of clusters to overlap in arbitrary ways is necessary to accommodate these two separate sources of similarity.

Figure 10 provides a graphical visualisation of the same additive clustering representation, as previously presented in [42]. A two-dimensional Euclidean multidimensional scaling representation of the numerals is overlayed with the derived clusters from the additive clustering representation. While the multidimensional scaling representation is clearly inferior to the additive clustering representation, in that it explains only about 60% of the variance, the spatial configuration it provides does allow the various additive clusters to be visualised easily. Unfortunately, as noted in [42], there is no guarantee that a multidimensional scaling representation will be sufficiently accurate to facilitate the drawing of relatively small convex clusters, as are evident in Figure 10. For strongly non-metric data, it is possible that the incompatibility between an additive clustering representation and its best-fitting two-dimensional spatial configuration will make the visualisation approach of Figure 10 relatively uninterpretable. When a suitable configuration is available, however, the generated displays would seem to be of some use in presenting an additive clustering representation, and it would be straightforward to include information relating to cluster weights through the use of labels, or by using cluster boundaries with different thicknesses.

6 Discussion

6.1 Choice of Representational Model

These demonstrations of multidimensional scaling, additive tree, and additive clustering representations highlight both their utility and their diversity. As a group, the three approaches constitute powerful but fundamentally different means of representing simi-

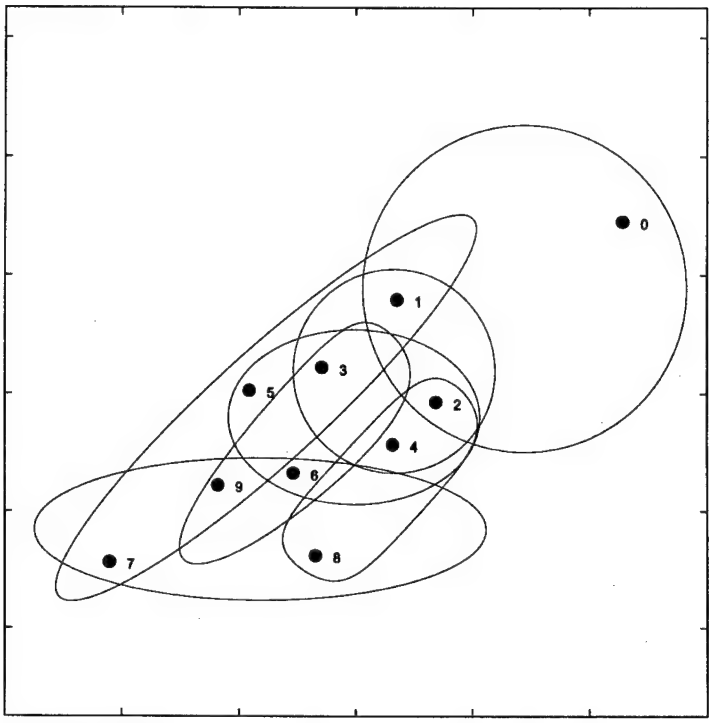

Figure 10: A graphical depiction of the additive clustering representation for the Arabic numerals, with the derived clusters overlayed upon the best-fitting two-dimensional spatial configuration generated by multidimensional scaling.

larity or proximity data. This means that it can be important to choose the appropriate representational model for a particular task. While it is not possible to be definitive in this regard, there are a number of observations that may be of some use.

The spatial concept of distance preservation that underpins multidimensional scaling representations often makes them well suited to modelling inherently continuous, spatial, and low-dimensional domains. In terms of psychological representational modelling, it has frequently been suggested [8, 72, 74] that multidimensional scaling representations are better suited to low-level and continuous perceptual domains, such as tones and colors, than they are to abstract conceptual or cognitive domains.

Where multidimensional scaling uses a strictly metric notion of distance preservation, additive trees are based on a more relaxed approach that might be described as 'topological'. In effect, additive trees supplement multidimensional scaling representations with a mechanism describing the patterns of transversal or transformation by which distance accrues across an otherwise homogeneous space. Accordingly, if the domain of interest is a low-dimensional and largely continuous one, but contains discontinuities, holes, or some other significant distortion within its representational space, then the use of an additive tree might prove appropriate.

It has also been argued [16] that "any set of objects that has arisen through an evolutionary process of 'splitting' or successive differentiation is likely to be modeled successfully by some sort of tree model", and that additive trees are particularly suited when the rates of change of these parallel processes differ significantly. On this basis, it seems reasonable to expect additive trees to provide useful models of similarity or proximity measures arising from developmental, evolutionary or longitudinal data.

Additive clustering is entirely unencumbered by metric constraints, and uses a discrete approach to representation that renders it appropriate for inherently featural domains. As the Arabic numeral example demonstrated, the ability of clusters to overlap in arbitrary ways allows additive clustering representations to capture multiple sources of similarity. For these reasons, it has been suggested [8, 72, 74] that this approach is well suited to modelling high-level, abstract or conceptual domains.

In general, these guidelines should be applied using prior knowledge of both the domain from which similarity or proximity data were collected, and the goals of the representational modelling. There has also been some work [57, 75] attempting to characterise the three representational approaches in terms of the distributions of similarity or proximity data, or a diagnostic measure calculated from these distributions, for which each is best suited. For the most part, however, these attempts have been of limited success, and probably could not be regarded as prescriptive in most realistic cases[12].

6.2 Alternative Algorithms

As was noted in the Introduction, each of the three representational models considered in this report has a long history within the field of cognitive modelling. Accordingly, a number of algorithms for fitting multidimensional scaling, additive tree, or additive

[12]Exceptions might be the diagnostic measures termed 'centrality' and 'reciprocity' [75]. When these measures assume high values, additive tree representations are often appropriate.

clustering models to similarity or proximity data have previously been proposed. It seems appropriate, therefore, to discuss those algorithms that might serve as alternatives to those developed and described here.

There are at least two specific multidimensional scaling algorithms that allow the use of distance metrics not subsumed by the weighted Minkowskian family, as required by the `mds.m` algorithm. One of these [17] allows multidimensional scaling representations to be generated on a sphere or a circle, so that the distance between points on either surface is measured by the geodesic distance between them. In a similar vein, [43] describes an algorithm that embeds multidimensional scaling representations on a Riemannian manifold of constant curvature. A related general approach to generating multidimensional scaling under constraints is presented in [4].

On a different front, there are a number of established *non-metric* multidimensional scaling algorithms [4, 18, 35, 63] that do not assume the same level of data scaling required by `mds.m`. Under the non-metric approach, the sum squared error measure is replaced by a criteria that measures only the ordinal, or rank, preservation of the assumed monotonically decreasing relationship between similarity and distance. That is, these algorithms generate a representation by attempting to ensure that the most similar object to a particular object is the nearest in the space, the second most similar is the second nearest, and so on in relation to all of the objects, without regard to the magnitude of the similarities and the spatial distances themselves. One of the remarkable properties of non-metric multidimensional scaling is that, particularly in low dimensional representational spaces, these ordinal constraints are sufficient to recover metric information. This is because the large number of ordinal constraints employed by non-metric multidimensional scaling often make it possible to determine coordinate locations for a set of objects with great accuracy. Sometimes, it may prove easier to produce reliable similarity or proximity data when only rank orderings are required, particularly when dealing with human judgments of preference. In these cases, the use of an alternative non-metric multidimensional scaling algorithm is likely to be appropriate.

There are also a number of alternative algorithms for generating additive trees, conveniently summarised in [16]. Many of these algorithms are based on a combinatorial approach, which builds an additive tree upwards from the leaves by successively combining objects using a heuristically guided search [14, 15, 57]. These algorithms are reasonably computationally efficient, and can be proven to find the optimal solution for proximity data that satisfies the additive inequality [6]. Unfortunately, most real proximity data significantly violates this fairly stringent set of constraints.

For this reason, the mathematical programming algorithm presented in [20] converts a proximity matrix that does not satisfy the additive inequality into one that does, while attempting to change the original matrix as little as possible. This modified matrix is then used as input to a combinatorial algorithm, which generates an additive tree representation. A second mathematical programming algorithm [9] is based on the observation [8, 57] that additive trees can be represented as the compositional sum of what is called an ultrametric tree [16], and a singular (or star) tree with one internal root node. Accordingly, the algorithm attempts to partition a proximity matrix into two parts, from which these two representational components are generated.

None of these alternative algorithms, however, allows for the systematic consideration

of additive tree representations with different topologies. In particular, none of these algorithms gives explicit control over the number of internal nodes used. As argued in [39], although each alternative algorithm is theoretically capable of generating an additive tree representation with less than the upper bound of $(n-2)$ internal nodes for an n object domain, they appear rarely to produce results significantly below this limit in practice. For this reason, unless the considerable computational demands of the `addtree.m` algorithm are excessive for a particular application, its provision of explicit complexity control may well make it preferable to the alternative algorithms.

The simple algorithm `displaytree.m` used to draw an additive tree representation is useful, but is obviously amenable to considerable refinement and extension. In particular, as the algorithm currently stands, it can take several attempts before a display as 'clean' as the one presented in Figure 8 is obtained. To overcome this deficiency, the algorithm could be augmented with the various 'aesthetic' layout constraints, such as seeking to avoid intersecting arcs, as used in several established graph-drawing algorithms [13]. This extension would be significantly aided by recent work evaluating the effectiveness of these constraints in facilitating human understanding [55].

Finally, it should be noted that there are number of alternative algorithms for generating additive clustering representations, at least three of which are worthy of some consideration. The widely used mathematical programming algorithm, MAPCLUS [1], as with its extentions and variants [10, 11, 24], uses complicated multi-stage optimisation routines together with a collection of *ad hoc* refinement processes. A very different additive clustering algorithm, based on qualitative factor analysis, is described in [47], and extracts clusters sequentially from a similarity matrix using various measures of cluster coherence and associated termination criteria. Although this algorithm is less widely used, those results that are available suggest that it is capable of generating good additive clustering representations. Perhaps the most impressive alternative additive clustering algorithm, however, is that described in [72]. This algorithm relies on a probabilistic formulation of additive clustering models that treats the cluster membership variables as 'hidden' or 'latent' variables, and optimises a maximum-likelihood performance measure using the Expectation-Maximization approach [22]. Besides being the outcome of an explicit and principled conceptual framework for additive clustering, the algorithm is also shown [72] to generate improved representational models for two well studied similarity matrices.

All of these alternative additive clustering algorithms, like the `adclus.m` algorithm, are computationally demanding. This means that the choice of algorithm is best based on the quality of the representations each generates. Where direct comparisons can be made, the qualitative factor analytic and probabilistic algorithms significantly outperform MAPCLUS. It is, therefore, somewhat promising to note that a limited comparative study [41] showed that a slight variant of the `adclus.m` algorithm matched or exceeded the performance of the probabilistic algorithm, and narrowly failed to achieve the performance of the qualitative factor analytic algorithm. Overall, however, the difference in the performance of the three algorithms appeared to be marginal, and each would seem capable of generating good additive clustering representations.

6.3 Conclusion

The ability of a domain model to facilitate understanding, provide explanatory insight, and allow for prediction and generalisation is determined largely by the quality of its underlying representation. A good representation characterises the structure of the domain as a whole, and captures its essence by abstracting the fundamental properties of the domain, and the nature of constraints under which it operates. Nowhere is the modelling imperative for generating sophisticated representations more acute than in cognitive modelling, where it has been argued that mental representational abstraction "is the essence of intelligence" [5], and that "pinning down mental representation is the route to rigor in psychology" [53]. Not surprisingly, therefore, there are a number of well developed cognitive representational models that may be of considerable use in the development of more general representational models.

This report developed and demonstrated techniques for building cognitive representations, using the different, and largely complementary, approaches of multidimensional scaling, additive trees, and additive clustering. Because each of these models is based on representing the similarities or proximities between sets of objects, a survey was provided of ways in which this type of data may be generated. Algorithms for finding multidimensional scaling, additive tree, and additive clustering representations were then described and demonstrated, together with a discussion suggesting when each should be used. In addition, an outline of the relative strengths and weaknesses of various previously developed alternative algorithms for generating these representations was presented. Finally, a principled Bayesian means for determining the appropriate complexity of each type of representation was proposed and demonstrated. This novel contribution enables the generation of sophisticated but simple representations that are only as complicated as the precision of the available data and the effectiveness of the representational model warrants. As Einstein evidently remarked [46]: "Everything should be made as simple as possible—but never simpler".

Acknowledgments

I wish to thank Garry Newsam for his detailed and thoughtful comments on several earlier drafts of this report. His suggestions and criticisms have greatly improved the final version. I also wish to thank Chris Woodruff for his contribution to this report, and to the earlier research papers summarised here. Finally, I wish to acknowledge that my understanding of a number of the model complexity issues addressed in the report has benefitted from communication with Greg Ashby, In Jae Myung and Kenneth Pope.

References

1. P Arabie and J D Carroll. MAPCLUS: A mathematical programming approach to fitting the ADCLUS model. *Psychometrika*, 45(2):211–235, 1980.

2. S Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie-Mellon University, 1994.

3. S Baluja. Genetic algorithms and explicit search statistics. In M C Mozer, M I Jordan, and T Petsche, editors, *Advances in Neural Information Processing Systems*, Cambridge, MA, 1996. MIT Press.

4. I Borg and J Lingoes. *Multidimensional similarity structure analysis.* Springer Verlag, New York, NY, 1987.

5. R A Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

6. P Buneman. The recovery of trees from measures of dissimilarity. In F R Hodson, D G Kendall, and P Tautu, editors, *Mathematics in the Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, Edinburgh, UK, 1971.

7. F Calliesz. The analytical solution of the additive constant problem. *Psychometrika*, 48:305–308, 1983.

8. J D Carroll. Spatial, non-spatial and hybrid models for scaling. *Psychometrika*, 41:439–463, 1976.

9. J D Carroll and S Pruzansky. Fitting of hierarchical tree structure (HTS) models, mixtures of HTS models, and hybrid models, via mathematical programming and alternating least squares, 1975. Paper presented at the U.S.-Japan seminar in theory, methods, and applications of multidimensional scaling and related techniques, San Diego, CA.

10. J.D. Carroll and P. Arabie. An individual differences generalization of the ADCLUS model and the MAPCLUS algorithm. *Psychometrika*, 48:157–169, 1983.

11. A Chaturvedi and J D Carroll. An alternating combinatorial optimization approach to fitting the INDCLUS and generalized INDCLUS models. *Journal of Classification*, 11:155–170, 1994.

12. J D Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions On Computer-Human Interaction*, 4(3):197–229, 1997.

13. M K Coleman and D S Parker. Aesthetics-based graph layout for human consumption. *Software–Practice and Experience*, 26(12):1415–1438., 1996.

14. J E Corter. ADDTREE/P: A PASCAL program for fitting additive trees based on Sattath and Tversky's ADDTREE algorithm. *Behavior Research Methods and Instrumentation*, 14:353–354, 1982.

15. J E Corter. A new combinatoric algorithm for fitting additive trees, 1996. Unpublished manuscript, Teachers College, Columbia University.

16. J E Corter. *Tree Models of Similarity and Association.* Sage, Thousand Oaks, CA, 1996.

17. T F Cox and M A A Cox. Multidimensional scaling on a sphere. *Communications in Statistics: Theory and Methods*, 20(9):2943–2953, 1991.

18. T F Cox and M A A Cox. *Multidimensional scaling.* Chapman and Hall, London, 1994.

19. M Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267:843–848, 1995.

20. G De Soete. A least squares algorithm for fitting additive trees to proximity data. *Psychometrika*, 48:621–626, 1983.

21. P Demartines and J Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.

22. A P Dempster, N M Laird, and D B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

23. J E Dennis, Jr. Nonlinear least squares. In D Jacobs, editor, *State of the Art in Numerical Analysis*, pages 269–312. Academic Press, New York, NY, 1977.

24. W S DeSarbo. GENNCLUS: New models for general nonhierarchical cluster analysis. *Psychometrika*, 47:449–475, 1982.

25. G Ekman. Dimensions of color vision. *The Journal of Psychology*, 38:467–474, 1954.

26. W R Garner. *The processing of information and structure.* Erlbaum, Potomac, MD, 1974.

27. I Gati and A Tversky. Representations of qualitative and quantitative dimensions. *Journal of Experimental Psychology: Human Perception and Performance*, 8(2):325–340, 1982.

28. F Girosi, M Jones, and T Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

29. D W Hamlyn. *The Penguin History of Western Philosophy.* Penguin, London, 1987.

30. J Hertz, A Krogh, and R G Palmer. *Introduction to the Theory of Neural Computing.* Addison-Wesley, Redwood City, CA, 1991.

31. G L Huba, J A Wingard, and P M Bentler. A comparison of two latent variable causal models for adolescent drug use. *Journal of Personality and Social Psychology*, 40(1):180–193, 1981.

32. E J Johnson and A Tversky. Representations of perceptions of risks. *Journal of Experimental Psychology: General*, 113(1):55–70, 1984.

33. R E Kass and A E Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

34. J K Kruschke. Human category learning: Implications for backpropagation models. *Connection Science*, 5:3–36, 1993.

35. J B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

36. C L Lawson and R J Hanson. *Solving Least Squares Problems.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

37. M D Lee. The connectionist construction of psychological spaces. *Connection Science*, 9(4):323–351, 1997.

38. M D Lee. Determining the dimensionality of multidimensional scaling representations for cognitive modeling. Manuscript submitted for publication, 1998.

39. M D Lee. Generating and displaying additive trees with limited complexity, 1998. Manuscript submitted for publication.

40. M D Lee. On the complexity of additive clustering models. Manuscript submitted for publication, 1998.

41. M D Lee. A simple method for learning optimal additive clustering models, 1998. Manuscript submitted for publication.

42. M D Lee and D Vickers. Psychological approaches to data visualisation. DSTO–RR–0135, 1998.

43. H Lindman and T Caelli. Constant curvature Riemannian scaling. *Journal of Mathematical Psychology*, 17:89–109, 1978.

44. D J C MacKay. *Information Theory, Pattern Recognition and Neural Networks.* [http://wol.ra.phy.cam.ac.uk/mackay/itprnn/course.html#book], Cambridge University, 1997.

45. G A Miller and P E Nicely. An analysis of perceptual confusions among some English consonants. *Journal of the Acoustical Society of America*, 27:338–352, 1955.

46. M Minsky. *The Society of Mind.* Simon & Schuster, New York, NY, 1986.

47. B G Mirkin. Additive clustering and qualitative factor analysis methods for similarity matrices. *Journal of Classification*, 4:7–31, 1987.

48. B G Mirkin. Erratum. *Journal of Classification*, 6:271–272, 1989.

49. J J More. The Levenberg-Marquardt algorithm: Implementation and theory. In G A Watson, editor, *Lecture Notes in Mathematics, 630*, pages 105–116. Springer-Verlag, 1977.

50. I J Myung and M A Pitt. Applying Occam's razor in modeling cognition: A Bayesian approach. *Psychonomic Bulletin & Review*, 4(1):79–95, 1997.

51. I Newton. *Opticks.* Smith and Walford, London, 1704. book 1, part 2, prop. 6.

52. R M Nosofsky. Similarity scaling and cognitive process models. *Annual Review of Psychology*, 43:25–53, 1992.

53. S Pinker. *How the Mind Works.* The Softback Preview, Great Britain, 1998.

54. W H Press, B P Flannery, S A Teukolsky, and W T Vetterling. *Numerical Recipes in C.* Cambridge University Press, New York, NY, 1988.

55. H C Purchase. Effects of graph layout. In *Proceedings of OZCHI 98*, pages 80–86, Los Alamitos, CA, 1998. IEEE Computer Society.

56. J Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, Singapore, 1989.

57. S Sattath and A Tversky. Additive similarity trees. *Psychometrika*, 42:319–345, 1977.

58. I J Schoenberg. Remarks to Maurice Fréchet's article "Sur la définition axiomatique d'une classe d'espaces vectoriels distanciés applicables vectoriellement sur l'espace de Hilbert". *Année Mathématique*, 36:724–732, 1935.

59. G Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

60. C E Shannon. The mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–243, 623–656, 1948.

61. R N Shepard. Stimulus and response generalization: A stochastic model relating generalization to distance in psychological space. *Psychometrika*, 22(4):325–345, 1957.

62. R N Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. II. *Psychometrika*, 27(3):219–246, 1962.

63. R N Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. I. *Psychometrika*, 27(2):125–140, 1962.

64. R N Shepard. Psychological representation of speech sounds. In E E David and P B Denes, editors, *Human communication: A unified view*, pages 67–113. McGraw Hill, New York, NY, 1972.

65. R N Shepard. Representation of structure in similarity data: Problems and prospects. *Psychometrika*, 39(4):373–422, 1974.

66. R N Shepard. Multidimensional scaling, tree-fitting, and clustering. *Science*, 210:390–398, 1980.

67. R N Shepard. Toward a universal law of generalization for psychological science. *Science*, 237:1317–1323, 1987.

68. R N Shepard. Integrality versus separability of stimulus dimensions: From an early convergence of evidence to a proposed theoretical basis. In J R Pomerantz and G L Lockhead, editors, *The perception of structure: Essays in honor of Wendell R Garner*, pages 53–71. American Psychological Association, Washington, DC, 1991.

69. R N Shepard. Perceptual-cognitive universals as reflections of the world. *Psychonomic Bulletin & Review*, 1(1):2–28, 1994.

70. R N Shepard and P Arabie. Additive clustering representations of similarities as combinations of discrete overlapping properties. *Psychological Review*, 86(2):87–123, 1979.

71. R N Shepard, D W Kilpatrick, and J P Cunningham. The internal representation of numbers. *Cognitive Psychology*, 7:82–138, 1975.

72. J B Tenenbaum. Learning the structure of similarity. In D S Touretzky, M C Mozer, and M E Hasselmo, editors, *Advances in neural information processing systems*, volume 8, pages 3–9. MIT Press, Cambridge, MA, 1996.

73. The Mathworks. Matlab, version 5.2., 1998.

74. A Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

75. A Tversky and J W Hutchinson. Nearest neighbor analysis of psychological spaces. *Psychological Review*, 93(1):3–22, 1986.

76. G Young and A S Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22, 1938.

77. R S Zemel. Minimum description length analysis. In M A Arbib, editor, *Handbook of Brain Theory and Neural Networks*, pages 572–575. MIT Press, Cambridge, MA, 1995.

Appendix A
Multidimensional Scaling Algorithms

File: mds.m

```
function [points,vaf]=mds(distance,dimensions,metric,iterations,learnrate)

% MDS multidimensional scaling (michael.d.lee@dsto.defence.gov.au)
% [points,vaf]=mds(distance,dimensions,metric,iterations,learnrate)
%
% DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
% DIMENSIONS specifies the required dimensionality of the coordinate representation (required)
% METRIC specifies the Minkowski distance metric operating in the space (default=2)
% ITERATIONS specifies the number of optimisation iterations performed (default=50)
% LEARNRATE specifies the learning rate used in optimisation (default=0.05)
%
% POINTS returns an NxDIMENSIONS matrix giving the derived coordinate locations
% VAF returns the variance of the distance values accounted for by the solution

% check the number of arguments
error(nargchk(2,5,nargin));

% check the distance matrix
[n check]=size(distance);
if check~=n
   error('distance matrix must be square');
end;
if ~isequal(distance,distance')
   error('distance matrix must be symmetric');
end;

% check the number of dimensions
if (dimensions<1)|(dimensions~=round(dimensions))
   error('number of dimensions must be a positive integer');
end;

% set default arguments as necessary
if nargin<5, iterations=50; end;
if nargin<4, learnrate=0.05; end;
if nargin<3, metric=2; end;

% check the number of iterations
if (iterations<1)|(iterations~=round(iterations))
   error('number of iterations must be a positive integer');
end;

% metric and learnrate are expected to be positive numbers
% but these constraints are not explicitly imposed

% assign shorter argument names
d=distance;
dim=dimensions;
r=metric;
lr=learnrate;

% normalise distances to lie between 0 and 1
reshift=min(min(d));
d=d-reshift;
rescale=max(max(d));
d=d/rescale;
```

```
% calculate the variance of  the distance matrix
dbar=(sum(sum(d))-trace(d))/n/(n-1);
temp=(d-dbar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% initialise variables
its=0;
p=rand(n,dim)*.01-.005;
dh=zeros(n);

% main loop for the given number of iterations
while (its<iterations)
   its=its+1;
   % randomly permute the objects to determine the order
   % in which they are pinned for this iteration
   pinning_order=randperm(n);
   for i=1:n
      m=pinning_order(i);
      % having pinned an object, move all of the other on each dimension
      % according to the learning rule
      for j=1:n
         if ~(m==j)
            dh(m,j)=norm(p(m,:)-p(j,:),r);
            dh(j,m)=dh(m,j);
            for k=1:dim
               p(j,k)=p(j,k)-lr*(dh(m,j)-d(m,j))*dh(m,j)^(1-r)*abs(p(j,k)-p(m,k))^(r-1)*sign(p(j,k)-p(m,k));
            end;
         end;
      end;
   end;
end;

% calculate the variance of  the distance matrix
dbar=(sum(sum(d))-trace(d))/n/(n-1);
temp=(d-dbar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% return the sum-squared error, variance accounted for
% and coordinate location of the final solution
sse=sum(sum((d-dh).^2))-trace((d-dh).^2);
vaf=1-sse/vard;
points=p*rescale+reshift;
```

File: mds2.m

```
function [points,vaf]=mds2(distance,dimensions,metric)

% MDS multidimensional scaling, version 2 (michael.d.lee@dsto.defence.gov.au)
% [points,vaf]=mds(distance,dimensions,metric)
%
% DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
% DIMENSIONS specifies the required dimensionality of the coordinate representation (required)
% METRIC specifies the Minkowski distance metric operating in the space (default=2)
%
% POINTS returns an NxDIMENSIONS matrix giving the derived coordinate locations
% VAF returns the variance of the distance values accounted for by the solution

global A flatd r;

% check the number of arguments
error(nargchk(2,3,nargin));

% check the distance matrix
```

```
[n check]=size(distance);
if check~=n
   error('distance matrix must be square');
end;
if ~isequal(distance,distance')
   error('distance matrix must be symmetric');
end;

% check the number of dimensions
if (dimensions<1)|(dimensions~=round(dimensions))
   error('number of dimensions must be a positive integer');
end;

% set default arguments as necessary
if nargin<3, metric=2; end;

% metric is expected to be a positive number
% but this constraints is not explicitly imposed

% assign shorter argument names
d=distance;
dim=dimensions;
r=metric;

% normalise distances to lie between 0 and 1
reshift=min(min(d));
d=d-reshift;
rescale=max(max(d));
d=d/rescale;

% calculate the variance of  the distance matrix
dbar=(sum(sum(d))-trace(d))/n/(n-1);
temp=(d-dbar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% initialise variables
options=zeros(18,1);
options(1)=-1; %display
options(2)=1e-1; %x precision
options(3)=1e-1; %residuals precision
npairs=round(n*(n-1)/2);

% setup differences between coordinates
A=zeros(npairs,n);
cc=1;
for i=1:n-1
   for j=i+1:n
      A(cc,i)=1;
      A(cc,j)=-1;
      cc=cc+1;
   end;
end;

% setup target distances as vector
flatd=zeros(npairs,1);
cc=1;
for i=1:n-1
   for j=i+1:n
      flatd(cc)=d(i,j);
      cc=cc+1;
   end;
end;

% minimise the residual function and find residual
p0=rand(n,dim)*.05;
p=leastsq('mdsresiduals',p0,options);
resid=mds_leastsq(p);
```

```
% return the variance accounted for
% and coordinate location of the final solution
sse=sum(sum(resid.^2));
vaf=1-sse/vard;
points=p*rescale+reshift;
```

File: mdsresiduals.m

```
function residuals = mds(x);
global A flatd r;
residuals=(sum(abs((A*x)).^r,2)).^(1/r)-flatd;
```

File: classicalmds.m

```
function [points,vaf]=classicalmds(distance,dimensions)

% Classical multidimensional scaling, (michael.d.lee@dsto.defence.gov.au)
% [points,vaf]=classicalmds(distance,dimensions)
%
% DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
% DIMENSIONS specifies the required dimensionality of the coordinate representation (required)
%
% POINTS returns an NxDIMENSIONS matrix giving the derived coordinate locations
% VAF returns the variance of the distance values accounted for by the solution

% check the number of arguments
error(nargchk(2,2,nargin));

% check the distance matrix
[n check]=size(distance);
if check~=n
   error('distance matrix must be square');
end;
if ~isequal(distance,distance')
   error('distance matrix must be symmetric');
end;

% check the number of dimensions
if (dimensions<1)|(dimensions~=round(dimensions))
   error('number of dimensions must be a positive integer');
end;

% assign shorter argument names
d=distance;
dim=dimensions;

% normalise distances to lie between 0 and 1
reshift=min(min(d));
d=d-reshift;
rescale=max(max(d));
d=d/rescale;

% calculate preliminary matrices
a=-.5*d.^2;
b=zeros(n);
for i=1:n
   for j=1:n
```

```
      b(i,j)=a(i,j)-sum(a(i,:))-sum(a(:,j))+sum(sum(a));
   end;
end;

% do ordered eigen-decomposition
[v e]=eig(b);
e=diag(e);
[val ind]=sort(e);
ind=flipud(ind);
e=diag(e(ind));
v=v(:,ind);

% return the variance accounted for
% and coordinate location of the final solution
dh=zeros(n);
for i=1:n-1;
   for j=i+1:n
      dh(i,j)=norm(v(i,1:dim)-v(j,1:dim));
   end;
end;
dh=dh+dh';
temp=corrcoef(reshape(d,n^2,1),reshape(dh,n^2,1));
vaf=temp(2,1)^2;
points=v(:,1:dim)*rescale+reshift;
```

Appendix B
Additive Tree Generation Algorithm

File: addtree.m

```
function [adjacency,lengths,vaf]=addtree(distance,nodes,learnrate,maxtrials,batchsize,batchlearn,mutprob,mutshift)

% ADDTREE additive tree (michael.d.lee@dsto.defence.gov.au)
% [adjacency,lengths,vaf]=addtree(distance,nodes,learnrate,maxtrials,batchsize,batchlearn,mutprob,mutshift)
%
% DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
% NODES specifies the number of internal nodes to place in the tree (required)
% LEARNRATE specifies the learning rate used in optimisation (default=0.1)
% MAXTRIALS specifies the number of trees evaluated without improvement before terminating (default=3000)
% BATCHSIZE specifies the size of the batch of potential tree solutions (default=50)
% BATCHLEARN specifies the number of best solutions in the batch used in learning (default=1)
% MUTPROB specifies the probability of 'mutating' the PBIL probability vector (default=0.02)
% MUTSHIFT specifies the extent of a 'mutation' shift (default=0.05)
%
% ADJACENCY returns an (N+NODES)x(N+NODES) adjacency matrix defining the tree topology
% LENGTHS returns a vector of length (N+NODES) containing the arc-lengths for the tree
% VAF returns the variance of the distance values accounted for by the solution

% check the number of arguments
error(nargchk(2,8,nargin));

% check the distance matrix
[n check]=size(distance);
if check~=n
   error('distance matrix must be square');
end;
if ~isequal(distance,distance')
   error('distance matrix must be symmetric');
end;

% check the number of internal nodes
if (nodes<2)|(nodes~=round(nodes))
   error('number of internal nodes must be an integer >= 2');
end;

% set default arguments as necessary
if nargin<8, mutshift=0.05; end;
if nargin<7, mutprob=0.02; end;
if nargin<6, batchlearn=1; end;
if nargin<5, batchsize=50; end;
if nargin<4, maxtrials=3000; end;
if nargin<3, learnrate=0.1; end;

% check the maximum number of trials
if (maxtrials<1)|(maxtrials~=round(maxtrials))
   error('maxtrials must be a positive integer');
end;

% check the batchsize
if (batchsize<1)|(batchsize~=round(batchsize))
   error('batchsize must be a positive integer');
end;

% check the number of batchsize used to learn
if (batchlearn<1)|(batchlearn>batchsize)|(batchlearn~=round(batchlearn))
   error('batchlearn must be a positive integer between 1 and batchsize');
```

```
end;

% learnrate, mutprob and mutshift are expected to be positive numbers
% but these constraints are not explicitly imposed

%rename variables
lr=learnrate;
d=distance;
m=nodes;

% normalise distances to lie between 0 and 1
reshift=min(min(d));
d=d-reshift;
rescale=max(max(d));
d=d/rescale;

% calculate the variance of  the distance matrix
dbar=(sum(sum(d))-trace(d))/n/(n-1);
temp=(d-dbar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% express the distance matrix as a column vector
flatd=[];
for i=1:n-1
   for j=i+1:n
      flatd=[flatd ;d(i,j)];
   end;
end;

% init other variables and constants
npairs=round(n*(n-1)/2);
besterr=vard;
updates=50; %controls how often an update message is displayed

% find length of inner-node vector and index its components
count=1;
start(1)=0;stop(1)=0;
start(2)=0;stop(2)=0;
for i=3:m
   start(i)=count;
   count=count+ceil(log2(i-1));
   stop(i)=count-1;
end;
count=count-1;

% initialise probability vectors
pprob=0.5*ones(count,1); %inner node links
numt=n*ceil(log2(m));
ptprob=0.5*ones(numt,1); %terminal node links

% main PBIL loop
tr=0;
while tr<maxtrials
   % initialise solution and evaluation storage
   svp=zeros(batchsize,count);
   svpt=zeros(batchsize,numt);
   evaluate=zeros(batchsize,1);
   % generate solutions
   for up=1:batchsize
      for j=1:count
         svp(up,j)=(rand<pprob(j));
      end;
      for j=1:numt
         svpt(up,j)=(rand<ptprob(j));
      end;
      p=svp(up,:);
      p=p';
```

```
pt=svpt(up,:);
pt=pt';

% now evaluate solutions ...
% first convert PBIL probability strings into adjacency matrix g
% interpret inner node tree from p
g=zeros(m+n);g(2,1)=1;
for i=3:m
   j=bin2dec(int2str(p(start(i):stop(i)))')+1;
   if j>=i
      j=1;
   end;
   g(i,j)=1;
end;

% interpret terminal nodes from pt
for i=1:n
   j=bin2dec(int2str(pt((i-1)*ceil(log2(m))+1:i*ceil(log2(m))))')+1;
   if j>m
      j=1;
   end;
   g(i+m,j)=1;
end;

% find edges in unique path between every pair
% find parent list of each terminal node
pl=zeros(n,m);
for i=1:n
   current=find(g(m+i,:)==1);
   pl(i,current)=1;
   while current~=1
      current=find(g(current,:)==1);
      pl(i,current)=1;
   end;
end;

% object pairs x weight matrix for non-negative least squares
wm=zeros(npairs,m+n-1);
c=0;
for i=1:n-1
   for j=i+1:n
      c=c+1;
      pairmeet=max(intersect(find(pl(i,:)==1),find(pl(j,:)==1)));
      current=m+i;
      wm(c,current)=1;
      while find(g(current,:)==1)~=pairmeet
         current=find(g(current,:)==1);
         wm(c,current)=1;
      end;
      current=m+j;
      wm(c,current)=1;
      while find(g(current,:)==1)~=pairmeet
         current=find(g(current,:)==1);
         wm(c,current)=1;
      end;
   end;
end;

% find arc-lengths and evaluate sum-squared error
w=nnls(wm,flatd);
evaluate(up)=sum((wm*w-flatd).^2);

% if solution is best one found, reset trial counter,
% save the solution and display a message
% otherwise increment the trial counter
if evaluate(up)<besterr
   tr=0;
```

```
            besterr=evaluate(up);
            bestvaf=1-besterr/vard;
            best=g;
            bestw=w;
            msg=sprintf('better tree found: accounts for %1.2f percent of the variance',bestvaf*100);
            disp(msg);
            pause(.001);
         else
            tr=tr+1;
         end;

         % periodically display the trial counter
         if (tr>0)&(mod(tr,updates)==0)
            msg=sprintf('%d trials have elapsed without improvement',tr);
            disp(msg);
            pause(.001);
         end;

      end; %for up

      % sort the batch of evaluated solutions
      [evaluate order]=sort(evaluate);
      svp=svp(order,:);
      svpt=svpt(order,:);

      % use the best mupdate with competitive learning to adjust probability vectors
      for i=batchlearn:-1:1
         pprob=(1-lr)*pprob+svp(i,:)'*lr;
         ptprob=(1-lr)*ptprob+svpt(i,:)'*lr;
      end;

      % stochastically 'mutate' probability vectors using mutshift and mutprob
      for i=1:count
         if rand<mutprob
            if pprob(i)>.5
               pprob(i)=pprob(i)*(1-mutshift);
            else
               pprob(i)=pprob(i)*(1-mutshift)+mutshift;
            end;
         end;
      end;

      for i=1:numt
         if rand<mutprob
            if ptprob(i)>.5
               ptprob(i)=ptprob(i)*(1-mutshift);
            else
               ptprob(i)=ptprob(i)*(1-mutshift)+mutshift;
            end;
         end;
      end;

   end; %while tr

   % return the best adjacency matrix, its associated arc-lengths
   % and percentage of variance it accounts for
   adjacency=best+best';
   lengths=bestw*rescale+reshift;
   vaf=bestvaf;
```

Appendix C
Additive Tree Display Algorithm

File: displaytree.m

```
function [points,corr]=displaytree(distance,adjacency,lengths,labels,characters,preserveweight,iterations,learnrate)

% DISPLAYTREE displays an additive tree (michael.d.lee@dsto.defence.gov.au)
% [points,corr]=displaytree(distance,adjacency,lengths,labels,characters,preserveweight,iterations,learnrate)
%
% DISTANCE is an NxN symmetric matrix of pairwise distances or proximities (required)
% ADJACENCY is an (N+NODES)x(N+NODES) adjacency matrix defining the tree topology, as returned by addtree
% LENGTHS returns a vector of length (N+NODES) containing the arc-lengths for the tree, as returned by addtree
% LABELS is a string array of containing N labels for the terminal nodes (required)
% CHARACTERS specifies how many characters to include in the display (default=0 displays the entire label)
% PRESERVEWEIGHT specifies the relative emphasis given to preserving lengths in the tree (default=10)
% ITERATIONS specifies the number of optimisation iterations performed (default=50)
% LEARNRATE specifies the learning rate used in optimisation (default=0.05)
%
% POINTS returns the coordinate locations of each of the internal and terminal nodes (optional)
% CORR returns the correlation between the specified lengths and the displayed lengths (optional)

% check the number of arguments
error(nargchk(4,8,nargin));

% check the distance matrix
[n check]=size(distance);
if check~=n
   error('distance matrix must be square');
end;
if ~isequal(distance,distance')
   error('distance matrix must be symmetric');
end;

% check the adjacency matrix
[tot check]=size(adjacency);
if check~=tot
   error('adjacency matrix must be square');
end;
if ~isequal(adjacency,adjacency')
   error('adjacency matrix must be symmetric');
end;
if (length(find(adjacency==1))+length(find(adjacency==0)))~=prod(size(adjacency))
   error('adjacency matrix must be contain only 0 and 1');
end;

% check the lengths
check=length(lengths);
if check~=tot
   error('number of arc lengths must match adjacency matrix');
end;
if sum(lengths>=0)<tot
   error('arc lengths must be positive');
end;

% check the labels
if ~isstr(labels)
   error('labels must be a string array');
end;
[check labelwidth]=size(labels);
if check~=n
   error('labels must contain one string for each object');
```

```
end;

% set default arguments as necessary
if nargin<8, learnrate=0.05; end;
if nargin<7, iterations=50; end;
if nargin<6, preserveweight=10; end;
if nargin<5, characters=0; end;

% check the number of characters to be displayed
if (characters<0)|(characters>labelwidth)|(characters~=round(characters))
   error('characters must be a non-negative integer');
end;

% check the number of iterations
if (iterations<1)|(iterations~=round(iterations))
   error('number of iterations must be a positive integer');
end;

% learnrate and preserverate are expected to be positive numbers
% but these constraints are not explicitly imposed

% assign shorter argument names
d=distance;
lr=learnrate;
w=lengths;
labs=labels;
labc=characters;
g=adjacency;

% normalise distances to lie between 0 and 1
reshift=min(min(d));
d=d-reshift;
rescale=max(max(d));
d=d/rescale;

% and keep the lengths on the same scale
w=w/rescale;

% infer number of internal nodes
m=tot-n;

% append distance matrix to include internal nodes
% and set a minimum separation of terminals
d=max(.05,d);
d=[zeros(m),zeros(m,n);zeros(n,m),d];
[x y]=find(tril(g)==1);
for i=1:length(x)
   d(x(i),y(i))=w(x(i));
   d(y(i),x(i))=w(x(i));
end;

% initialise variables
its=0;
p=rand(m+n,2)*.01-.005;
dh=zeros(m+n);
% main optimisation loop
while (its<iterations)
   its=its+1;
   % select pinning order
   r=randperm(m+n);
   for i=1:m+n
      cc=r(i);
      for j=1:m+n
         if ~(cc==j|d(cc,j)==0)
            % update estimated distance matrix
            dh(cc,j)=norm(p(cc,:)-p(j,:));
            dh(j,cc)=dh(cc,j);
```

```
            % if not terminal-terminal pair keep the learning rate
            if ~((cc>m)&(j>m))
               lrt=lr;
               % otherwise lower it according to preserverate
            else
               lrt=1/preserveweight*lr;
            end;
            % update display locations
            for k=1:2
               p(j,k)=p(j,k)-lrt*(dh(cc,j)-d(cc,j))/dh(cc,j)*(p(j,k)-p(cc,k));
            end;
         end;
      end;
   end;
end;

% rescale then centre the coordinates
p=p*rescale+reshift;
p(:,1)=p(:,1)-p(1,1);
p(:,2)=p(:,2)-p(1,2);

% return points and correlation, as required
if nargout>1
   % check and checkh hold the actual and displayed arc lengths
   % used to calculate the corr measure
   check=[];
   checkh=[];
   for i=1:m+n
      for j=1:m+n
         if ~(i==j|d(i,j)==0|((i>m)&(j>m)))
            check=[check;d(i,j)];
            checkh=[checkh;dh(i,j)];
         end;
      end;
   end;
   temp=corrcoef(check,checkh);
   corr=temp(1,2);
end;
if nargout>0, points=p; end;

% draw the tree
% set figure window
figure(1);clf;hold on;

% draw arcs
for i=1:m+n
   j=find(g(i,:)==1);
   j=j(1);
   plot([p(i,1) p(j,1)],[p(i,2) p(j,2)],'k-','linewidth',1);
end;

% draw internal nodes
for i=1:m
   plot(p(i,1),p(i,2),'ko','markersize',8,'linewidth',2,...
      'markerfacecolor',[1 1 1],'markeredgecolor',[0 0 0]);
end;

% draw terminal nodes
% may want to adjust the markersize for some labels
for i=1:n
   plot(p(i+m,1),p(i+m,2),'ko','markersize',12+labc*4,'linewidth',1,...
      'markerfacecolor',[1 1 1],'markeredgecolor',[0 0 0]);
end;

% label terminal nodes
for i=1:n
   if labc==0
```

```
      text(p(i+m,1),p(i+m,2),deblank(labs(i,:)),...
         'horizontalalignment','center',...
         'fontname','arial','fontsize',8,'fontweight','bold');
   else
      text(p(i+m,1),p(i+m,2),deblank(labs(i,1:labc)),...
         'horizontalalignment','center',...
         'fontname','arial','fontsize',8,'fontweight','normal');
   end;
end;

% format axis and figure border
set(gca,'box','on','xticklabel','none','xtick',[],...
   'yticklabel','none','ytick',[]);
axis square;
axis(1.1*axis);

% display long label names if characters>0
if labc>0
text(max(get(gca,'xlim'))*1.1,max(get(gca,'ylim'))*.9,[upper(labs(:,1:labc)) labs(:,labc+1:end)],...
   'verticalalignment','top');
end;
```

Appendix D
Additive Clustering Algorithm

File: adclus.m

```
function [clusters,weights,vaf]=adclus(similarity,numberclusters,learnrate,maxtrials,batchsize,batchlearn,mutprob,mu

% ADCLUS additive clustering (michael.d.lee@dsto.defence.gov.au)
% [clusters,weights,vaf]=adclus(similarity,numberclusters,learnrate,maxtrials,batchsize,batchlearn,mutprob,mutshift)
%
% SIMILARITY is an NxN symmetric matrix of pairwise similarities (required)
% NUMBERCLUSTERS specifies the number of clusters to use (required)
% LEARNRATE specifies the learning rate used in optimisation (default=0.1)
% MAXTRIALS specifies the number of trees evaluated without improvement before terminating (default=3000)
% BATCHSIZE specifies the size of the batch of potential tree solutions (default=50)
% BATCHLEARN specifies the number of best solutions in the batch used in learning (default=1)
% MUTPROB specifies the probability of 'mutating' the PBIL probability vector (default=0.02)
% MUTSHIFT specifies the extent of a 'mutation' shift (default=0.05)
%
% CLUSTERS returns an Nx(NUMBERCLUSTERS+1) matrix defining derived cluster membership plus the universal cluster
% WEIGHTS returns a vector of length (NUMBERCLUSTERS+1) containing the weights of the clusters
% VAF returns the variance of the similarity values accounted for by the solution

% check the number of arguments
error(nargchk(2,8,nargin));

% check the similarity matrix
[n check]=size(similarity);
if check~=n
   error('similarity matrix must be square');
end;
if ~isequal(similarity,similarity')
   error('similarity matrix must be symmetric');
end;

% check the number of clusters
if (numberclusters<1)|(numberclusters~=round(numberclusters))
   error('number of clusters must be an integer >= 2');
end;

% set default arguments as necessary
if nargin<8, mutshift=0.05; end;
if nargin<7, mutprob=0.02; end;
if nargin<6, batchlearn=1; end;
if nargin<5, batchsize=50; end;
if nargin<4, maxtrials=3000; end;
if nargin<3, learnrate=0.1; end;

% check the maximum number of trials
if (maxtrials<1)|(maxtrials~=round(maxtrials))
   error('maxtrials must be a positive integer');
end;

% check the batchsize
if (batchsize<1)|(batchsize~=round(batchsize))
   error('batchsize must be a positive integer');
end;

% check the number of batchsize used to learn
if (batchlearn<1)|(batchlearn>batchsize)|(batchlearn~=round(batchlearn))
   error('batchlearn must be a positive integer between 1 and batchsize');
```

```
end;

% learnrate, mutprob and mutshift are expected to be positive numbers
% but these constraints are not explicitly imposed

%rename variables
lr=learnrate;
s=similarity;
m=numberclusters;

% normalise similarities to lie between 0 and 1
reshift=min(min(s));
s=s-reshift;
rescale=max(max(s));
s=s/rescale;

% calculate the variance of  the similarity matrix
sbar=(sum(sum(s))-trace(s))/n/(n-1);
temp=(s-sbar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% express the similarity matrix as a column vector
flats=[];
for i=1:n-1
   for j=i+1:n
      flats=[flats;s(i,j)];
   end;
end;

% init other variables and constants
npairs=round(n*(n-1)/2);
besterr=vard;
veclength=n*m;
p=.5*ones(veclength,1);
updates=50; %controls how often an update message is displayed

% main PBIL loop
tr=0;
while tr<maxtrials
   % initialise solution and evaluation storage
   sv=zeros(batchsize,veclength);
   evaluate=zeros(batchsize,1);
   % generate solutions
   for i=1:batchsize
      for j=1:veclength
         sv(i,j)=(rand<p(j));
      end;
      f=reshape(sv(i,:),n,m);
      % augment universal cluster for additive constant
      f=[f ones(n,1)];
      % form of cluster membership needed for non-negative least squares
      flatf=zeros(npairs,m+1);
      count=0;
      for x=1:n-1
         for y=x+1:n
            count=count+1;
            flatf(count,:)=f(x,:).*f(y,:);
         end;
      end;
      % find weights
      w=diag(nnls(flatf,flats));
      % evaluate solution
      sh=f*w*f';
      se=(s-sh).^2;
      evaluate(i)=.5*(sum(sum(se))-trace(se));

      % if solution is best one found, reset trial counter,
```

```
      % save the solution and display a message
      % otherwise increment the trial counter
      if evaluate(i)<besterr
         tr=0;
         besterr=evaluate(i);
         bestvaf=1-besterr/vard;
         best=f;
         bestw=diag(w);
         msg=sprintf('better clustering found: accounts for %1.2f percent of the variance',bestvaf*100);
         disp(msg);
         pause(.001);
      else
         tr=tr+1;
      end;

      % periodically display the trial counter
      if (tr>0)&(mod(tr,updates)==0)
         msg=sprintf('%d trials have elapsed without improvement',tr);
         disp(msg);
         pause(.001);
      end;
   end; %for i

   % sort the batch of evaluated solutions
   [evaluate order]=sort(evaluate);
   sv=sv(order,:);

   % use the best mupdate with competitive learning to adjust probability vectors
   for i=batchlearn:-1:1
      p=(1-lr)*p+sv(i,:)'*lr;
   end;

   % stochastically 'mutate' probability vector using mutshift and mutprob
   for i=1:veclength
      if rand<mutprob
         if p(i)>.5
            p(i)=p(i)*(1-mutshift);
         else
            p(i)=p(i)*(1-mutshift)+mutshift;
         end;
      end;
   end;

end; %while tr

% return the best cluster membership matrix, its associated weights
% and percentage of variance it accounts for
clusters=best;
weights=bestw*rescale+reshift;
vaf=bestvaf;
```

Appendix E
Bayesian Information Criterion Algorithm

File: bic.m

```
function value=bic(bictype,data,vaf,sigma,dims_nodes_clusters)

% BIC bayesian information criterion (michael.d.lee@dsto.defence.gov.au)
% value=bic(bictype,data,vaf,sigma,dims_nodes_clusters)
%
% BICTYPE specifies the model type using the string 'mds', 'addtree', or 'adclus' (required)
% DATA is the NxN symmetric matrix of similarities or proximities (required)
% VAF is variance of the similarity values accounted for by the representation (required)
% SIGMA is the assumed level of data precision (required)
% DIMS_NODES_CLUSTERS gives the number of dimensions, internal nodes, or clusters in the representation (required)
%
% VALUE returns the bayesian information criterion value

% check the number of arguments
error(nargchk(5,5,nargin));

% check the type
validtypes=char('mds','addtree','adclus');
if isempty(strmatch(bictype,validtypes,'exact'))
   error('bictype must be specified as ''mds'', ''addtree'', or ''adclus''')
end;

% check the similarity or proximity matrix
[n check]=size(data);
if check~=n
   error('similarity or proximity matrix must be square');
end;
if ~isequal(data,data')
   error('similarity or proximity matrix must be symmetric');
end;

% check the vaf measure
if (vaf<0)|(vaf>1)
   error('vaf should be between 0 and 1');
end;

% check the sigma value
if (sigma<0)
   error('sigma should be positive');
end;

% check the dims, nodes, or clusters
if (dims_nodes_clusters<1)|(dims_nodes_clusters~=round(dims_nodes_clusters))
   error('dims, nodes, or clusters must be a positive integer');
end;

% rename variable
m=dims_nodes_clusters;

% calculate the variance of  the data matrix
bar=(sum(sum(data))-trace(data))/n/(n-1);
temp=(data-bar*ones(n)).^2;
vard=.5*(sum(sum(temp))-trace(temp));

% calculate the sum squared error
sse=(1-vaf)*vard;
```

```
switch bictype
case 'mds', value=sse/sigma^2+m*n*log(n*(n-1)/2);
case 'addtree', value=sse/sigma^2+(m+n-1)*log(n*(n-1)/2);
case 'adclus', value=sse/sigma^2+(m+1)*log(n*(n-1)/2);
end;
```

DISTRIBUTION LIST

Algorithms for Representing Similarity Data

Michael D. Lee

	Number of Copies
DEFENCE ORGANISATION	
S&T Program	
Chief Defence Scientist FAS Science Policy AS Science Corporate Management	1
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Data Sht
Counsellor, Defence Science, Washington	Doc Data Sht
Scientific Adviser to MRDC, Thailand	Doc Data Sht
Scientific Adviser Policy and Command	1
Navy Scientific Adviser	Doc Data Sht
Scientific Adviser, Army	Doc Data Sht
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical and Maritime Research Laboratory	
Director, Aeronautical and Maritime Research Laboratory	Doc Data Sht
Chief, Air Operations Division	1
Chief, Maritime Operations Division	1
Research Leader, Human Factors, AOD	1
Research Leader, Land Weapons Systems, WSD	1
Research Leader, Sonar Technology and Processing, MOD	1
Dr. Ross Dawe, MOD	1
Electronics and Surveillance Research Laboratory	
Director, Electronics and Surveillance Research Laboratory	1
Chief, Communications Divison	1
Chief, Information Technology Division	1
Chief, Land Operations Division	1
Research Leader, Military Information Networks, CD	1
Research Leader, Command Control and Intelligence Systems, ITD	1
Research Leader, Human Factors, LOD	1
Head, Intelligent Networks, CD	1
Head, C3I Networks, CD	1

Head, Human Systems Integration, ITD	1
Head, Command and Control Australian Theatre, ITD	1
Head, Information Management and Fusion, ITD	1
Head, Image Analysis and Exploitation, SSD	1
Dr. Michael Lee (Author), CD	5
DSTO Libraries	
Library Fishermens Bend	1
Library Maribyrnong	1
Library Salisbury	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Data Sht
Capability Development Division	
Director General Maritime Development	Doc Data Sht
Director General Land Development	Doc Data Sht
Director General Aerospace Development	Doc Data Sht
Director General C3I Development	Doc Data Sht
Army	
ABCA Office, G-1-34, Russell Offices, Canberra	4
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool Military Area, NSW 2174	Doc Data Sht
Intelligence Program	
DGSCD, Defence Signals Directorate	1
Defence Intelligence Organisation	1
Corporate Support Program (libraries)	
Officer in Charge, TRS, Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre	Doc Data Sht
Additional copies for DEC for exchange agreements	
US Defense Technical Information Center	2
UK Defence Research Information Centre	2
Canada Defence Scientific Information Service	1
NZ Defence Information Centre	1
National Library of Australia	1
UNIVERSITIES AND COLLEGES	
Australian Defence Force Academy Library	1
Head of Aerospace and Mechanical Engineering, ADFA	1

Deakin University Library, Serials Section (M List)	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1
Librarian, Barr-Smith Library, University of Adelaide	1
Key Centre in Applied Cognition, University of Queensland	1

OTHER ORGANISATIONS

NASA (Canberra)	1
Australian Government Publishing Service	1
The State Library of South Australia	1
Parliamentary Library of South Australia	1

ABSTRACTING AND INFORMATION ORGANISATIONS

INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Engineering Societies Library, US	1
Materials Information, Cambridge Science Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK	1
Library – Exchange Desk, National Institute of Standards and Technology, US	1

SPARES

DSTO Salisbury Research Library	10

Total number of copies:	75

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. CAVEAT/PRIVACY MARKING
2. TITLE Algorithms for Representing Similarity Data			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)	
4. AUTHOR(S) Michael D. Lee			5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury, South Australia, Australia 5108	
6a. DSTO NUMBER DSTO–RR–0152	6b. AR NUMBER AR–010–971		6c. TYPE OF REPORT Research Report	7. DOCUMENT DATE May, 1999
8. FILE NUMBER E 8709–7–11 2	9. TASK NUMBER 98/126	10. SPONSOR DSD	11. No OF PAGES 58	12. No OF REFS 77
13. DOWNGRADING / DELIMITING INSTRUCTIONS Not Applicable			14. RELEASE AUTHORITY Chief, Communications Division	

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved For Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA, ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DEFTEST DESCRIPTORS

Cognitive Science
Scaling
Algorithms
Modelling
Clustering

19. ABSTRACT

This report develops and demonstrates algorithms for representing and displaying similarity data using three established cognitive models. The first representational model, multidimensional scaling, represents objects as points in a coordinate space so that similar objects lie near each other. The second representational model, the additive tree, represents objects as terminal nodes in a tree so that the similarity of two objects is modelled by length of the path between them. The third representational model, additive clustering, specifies a number of clusters with associated weights, so that the similarity of two objects is modelled by the sum of the weights of their common clusters. As well as listing and demonstrating MATLAB algorithms for finding these representations, a survey is presented of ways in which similarity and proximity data may be generated, and a principled Bayesian method of controlling the complexity of each representational model is presented. Finally, a number of suggestions are made regarding the use of the three representational models, and the relative strengths and weaknesses of the algorithms in relation to previously developed alternative algorithms are discussed.